Session 3

In C#, both Named Parameters and Positional Parameters are ways to pass arguments to methods or constructors. They offer different approaches for providing values to parameters based on the specific needs of the code. Here's an explanation of both concepts:

Positional Parameters:

Positional parameters are the default way of passing arguments in C#. When using positional parameters, you provide arguments in the same order as the parameters are defined in the method or constructor signature. The values are assigned to parameters based on their respective positions.

Named parameters allow you to explicitly specify the parameter name along with the argument value. This approach allows you to pass arguments in any order, as long as you specify the corresponding parameter names. It provides increased clarity and avoids ambiguity when dealing with methods or constructors that have multiple parameters of the same type.

In C#, the params keyword allows you to define a method parameter that can accept a variable number of arguments of a specified type. This feature is known as the "params parameter" or "variable-length parameter list". It enables you to pass a variable number of arguments to a method, making the method more flexible and convenient to use.

Local functions in C# are functions that are defined within the body of another method. They are similar to regular methods but are scoped within the containing method and can only be called from within that method. Local functions provide a way to encapsulate logic within a method, improving code organization and readability**.**

I**n C#, a readonly** property is a property that can only be assigned a value during object initialization or within the constructor. Once assigned, the value of a readonly property cannot be changed for the lifetime of the object. This provides a way to create properties that have a fixed value, ensuring their immutability.

In C#, you can create a read-only property using property accessors by providing a getter without a setter. By omitting the setter, you make the property read-only, meaning it can only be accessed and not modified.

Object initializer:

- Object initializer is a syntax in C# that allows initializing properties or fields of an object at creation.
- It uses curly braces to enclose a list of assignments for properties or fields, separated by commas.
- Object initializer simplifies initialization by providing a concise syntax, avoiding the need for explicit property setters or lengthy constructors.

Session 3

- It is particularly useful when creating instances with multiple properties or fields, improving code readability.
- The compiler generates the necessary assignments based on the object initializer syntax, simplifying the initialization process.

IDisposable:

- IDisposable is an interface in C# used for explicit resource management and cleanup.
- It defines a single method Dispose() that is responsible for releasing unmanaged resources held by an object.
- Implementing IDisposable allows for deterministic disposal of resources, ensuring timely cleanup and avoiding resource leaks.
- The Dispose() method is typically called explicitly or using the using statement to ensure proper resource cleanup.
- Proper implementation of IDisposable is essential for objects that consume limited or external resources such as file handles, database connections, or network sockets.

  In C#, a destructor, also known as a finalizer, is a special method that is automatically called when an object is being garbage collected. Its purpose is to perform cleanup tasks and release unmanaged resources held by the object before it is removed from memory.

  Here are a few key points about destructors:

  Syntax: A destructor is defined using the tilde (~) followed by the class name. It has no return type and takes no parameters.

  Execution: Destructors are automatically invoked by the garbage collector when it determines that an object is no longer reachable or when the program terminates. The exact timing of destructor execution is non-deterministic.

  Cleanup Tasks: Destructors are typically used to release unmanaged resources such as file handles, database connections, or network sockets. They provide a mechanism to ensure that these resources are properly released before the object is garbage collected.

  Limited Usage: Destructors are less commonly used in C# due to the presence of the IDisposable interface and the Dispose pattern, which provide a more controlled and deterministic way of resource cleanup. Destructors should be used sparingly, and explicit resource cleanup should be preferred.