Namespace

A **namespace** in programming is a container that provides context for the identifiers (names of types, functions, variables, etc.) it holds. By grouping related elements, namespaces prevent name conflicts and help organize code logically.

Namespaces in .NET

In the .NET ecosystem, namespaces are fundamental. They:

1. **Organize Classes and Types**: Help structure code into logical groupings.
2. **Avoid Naming Conflicts**: Prevent ambiguity when multiple libraries define the same class or method names.
3. **Simplify Code Reusability**: Allow developers to reference specific parts of a library without importing unrelated parts.

Defining a Namespace

In C#, a namespace is declared using the namespace

Common .NET Namespaces

- System: The core namespace containing basic types like Console, String, and collections.
- System.Collections.Generic: For generic collections such as List<T> and Dictionary<TKey, TValue>.
- System.IO: Provides classes for input/output operations (e.g., File, StreamReader).
- System.Linq: Enables LINQ queries for collections.
- System.Net: Contains classes for network programming like HttpClient.

And to reference a class or type from another namespace, developers use the using directive.

Nested Namespaces

- Namespaces can be nested to create hierarchical structures
- This approach improves code organization, especially in larger projects.

Global Namespace

In .NET, all namespaces exist under the implicit **global namespace**, which is the root level. A class declared without a namespace belongs to this global namespace:

```
class GlobalClass { }
```

Best Practices for Namespaces

1. **Logical Grouping**: Organize types into namespaces that reflect their functionality.
2. **Avoid Overlapping Names**: Ensure namespaces are unique across projects.
3. **Use Pascal Case**: Follow conventions like CompanyName.ProductName.Module.