Session: 10

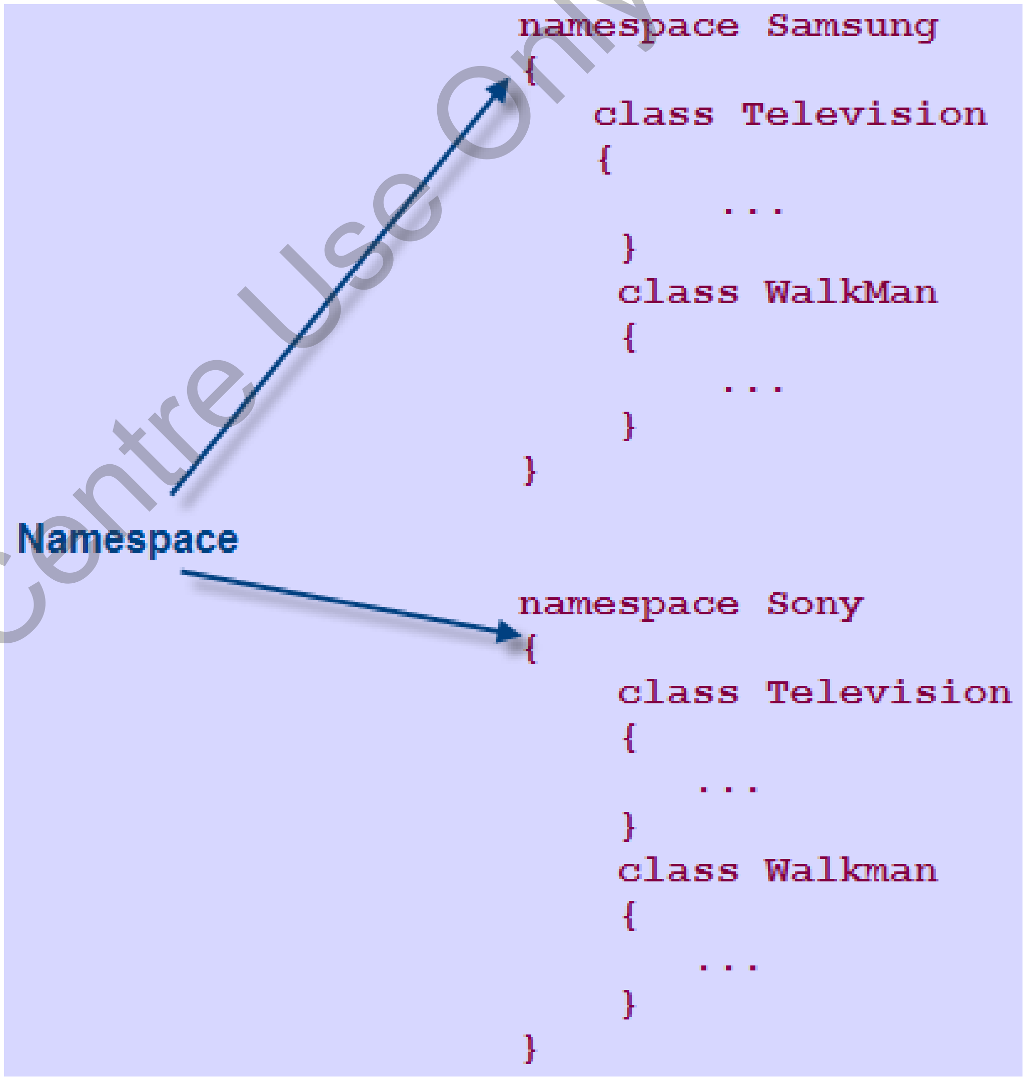
Namespaces

Objectives

- Define and describe namespaces
- Explain nested namespaces

Namespaces

- A namespace:
  - Is used in C# to group classes logically and prevent name clashes between classes with identical names.
  - Reduces any complexities when the same program is required in another application.



```
namespace Samsung
{
   class Television
   {
          . . .
    }
    class WalkMan
    {
          . . .
    }
}

namespace Sony
{
    class Television
    {
        . . .
    }
    class Walkman
    {
        . . .
    }
}
```

Introduction to Namespaces – Purpose 1-3

Example

- Consider Venice, which is a city in the US as well as in Italy.
- You can easily distinguish between the two cities by associating them with their respective countries.
- Similarly, when working on a huge project, there may be situations where classes have identical names.
- This may result in name conflicts.
- This problem can be solved by having the individual modules of the project use separate namespaces to store their respective classes.
- By doing this, classes can have identical names without any resultant name clashes.

Introduction to Namespaces – Purpose 2-3

- The following code renames identical classes by inserting a descriptive prefix:

Snippet

```
class SamsungTelevision
{
...
}
class SamsungWalkMan
{
...
}
class SonyTelevision
{
...
}
class SonyWalkMan
{
...
}
```

- In the code:
  - The identical classes **`Television`** and **`WalkMan`** are prefixed with their respective company names to avoid any conflicts.
  - There cannot be two classes with the same name.
  - It is observed that the names of the classes get long and become difficult to maintain.

Introduction to Namespaces – Purpose 3-3

- The following code demonstrates a solution to overcome this, by using namespaces:

Snippet

```
namespace Samsung
{
     class Television
     {
     ...
     }
     class WalkMan
     {
     ...
     }
}
namespace Sony
{
    class Television
     {
     ...
     }
     class Walkman
     {
     ...
     }
}
```

- In the code:
  - Each of the identical classes is placed in their respective namespaces, which denote respective company names.
  - It can be observed that this is a neater, better organized, and more structured way to handle naming conflicts.

Using Namespaces

- C# allows you to specify a unique identifier for each namespace.
- This identifier helps you to access the classes within the namespace.
- Apart from classes, the following data structures can be declared in a namespace:

Interface	•An interface is a reference type that contains declarations of the events, indexers, methods, and properties. •Interfaces are inherited by classes and structures and all the declarations are implemented in these classes and structures.
Structure	•A structure is a value type that can hold values of different data types. •It can include fields, methods, constants, constructors, properties, indexers, operators, and other structures.
Enumeration	•An enumeration is a value type that consists of a list of named constants. •This list of named constants is known as the enumerator list.
Delegate	•A delegate is a user-defined reference type that refers to one or more methods. •It can be used to pass data as parameters to methods.

Characteristics and Benefits

- A namespace groups common and related classes, structures, or interfaces, which support OOP concepts of encapsulation and abstraction.
- A namespace:
  - Provides a hierarchical structure that helps to identify the logic for grouping the classes.
  - Allows you to add more classes, structures, enumerations, delegates, and interfaces once the namespace is declared.
  - Includes classes with names that are unique within the namespace.
- A namespace provides the following benefits:
  - A namespace allows you to use multiple classes with same names by creating them in different namespaces.
  - It makes the system modular.

Built-in Namespaces 1-3

- The .NET Framework comprises several built-in namespaces that contain:
  - Classes
  - Interfaces
  - Structures
  - Delegates
  - Enumerations
- These namespaces are referred to as system-defined namespaces.
- The most commonly used built-in namespace of the .NET Framework is `System`.
- The `System` namespace contains classes that:
  - Define value and reference data types, interfaces, and other namespaces.
  - Allow you to interact with the system, including the standard input and output devices.

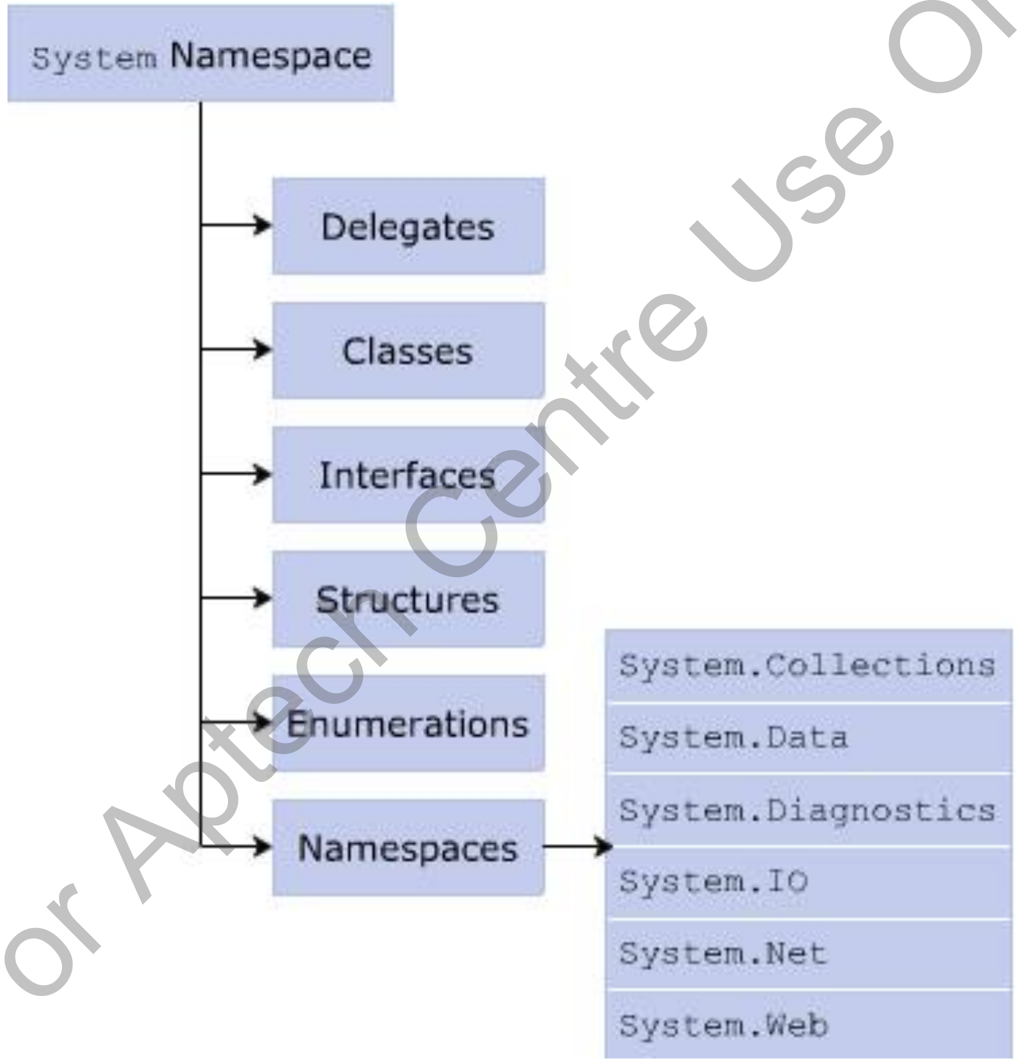
Built-in Namespaces 2-3

- Some of the most widely used namespaces within the `System` namespace are as follows:

`System.Collections`	• The `System.Collections` namespace contains classes and interfaces that define complex data structures such as lists, queues, bit arrays, hash tables, and dictionaries.
`System.Data`	• The `System.Data` namespace contains classes that make up the ADO.NET architecture. • The ADO.NET architecture allows you to build components that can be used to insert, modify, and delete data from multiple data sources.
`System.Diagnostics`	• The `System.Diagnostics` namespace contains classes that are used to interact with the system processes. • This namespace also provides classes that are used to debug applications and trace the execution of the code.
`System.IO`	• The `System.IO` namespace contains classes that enable you to read from and write to data streams and files.
`System.Net`	• The `System.Net` namespace contains classes that allow you to create Web-based applications.
`System.Web`	• The `System.Web` namespace provides classes and interfaces that allow communication between the browser and the server.

Built-in Namespaces 3-3

- The following figure displays some built-in namespaces:

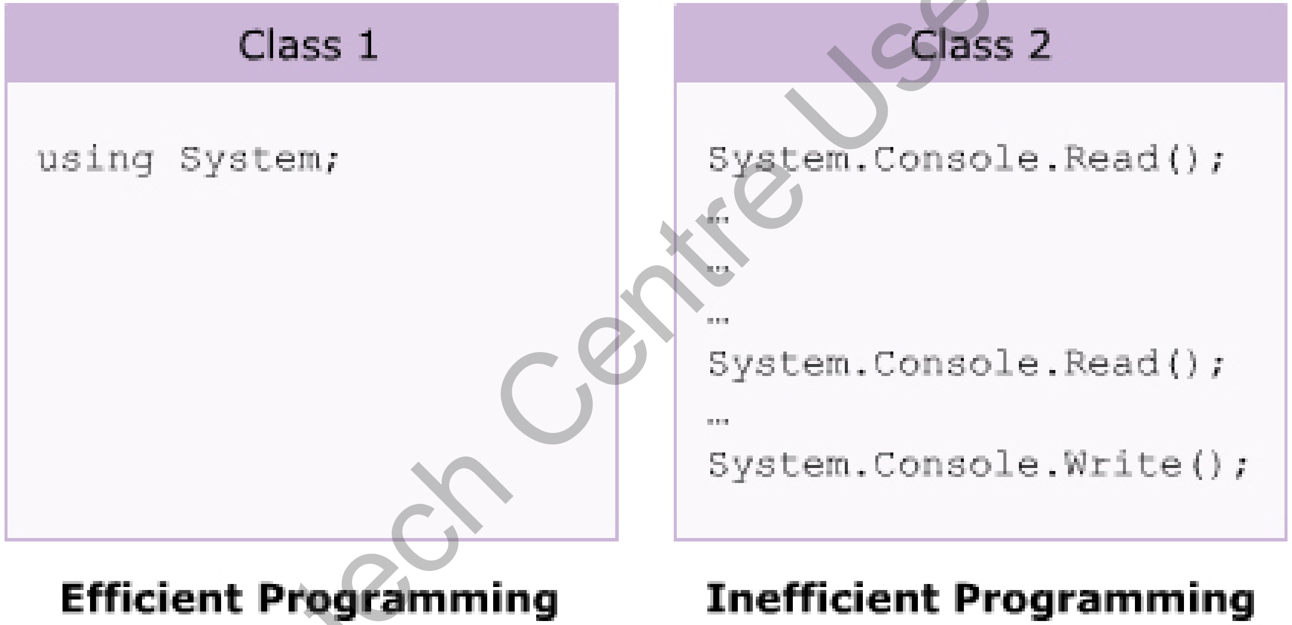


Using the System Namespace 1-6

- The `System` namespace is imported by default in the .NET Framework.
- It appears as the first line of the program along with the `using` keyword.
- For referring to classes within a built-in namespace, you need to explicitly refer to the required classes.
- It is done by specifying the namespace and the class name separated by the dot (`.`) operator after the `using` keyword at the beginning of the program.
- You can refer to classes within the namespaces in the same manner without the `using` keyword.
- However, this results in redundancy because you need to mention the whole declaration every time you refer to the class in the code.

Using the System Namespace 2-6

- The two approaches of referencing the `System` namespace are:



```
using System;
```

```
System.Console.Read();
…
…
…
System.Console.Read();
…
System.Console.Write();
```

- Though both are technically valid, the first approach is more recommended.

Using the System Namespace 3-6

- The following syntax is used to access a method in a system-defined namespace:

Syntax

```
<NamespaceName>.<ClassName>.<MethodName>;
```

- In the syntax:
  - `NamespaceName`: Is the name of the namespace.
  - `ClassName`: Is the name of the class that you want to access.
  - `MethodName`: Is the name of the method within the class that is to be invoked.

Using the System Namespace 4-6

- The following syntax is used to access the system-defined namespaces with the `using` keyword:

Syntax

```
using <NamespaceName>;
using <NamespaceName>.<ClassName>;
```

- In the syntax:
  - `NamespaceName`: Is the name of the namespace and it will refer to all classes, interfaces, structures, and enumerations.
  - `ClassName`: Is the name of the specific class defined in the namespace that you want to access.

Using the System Namespace 5-6

- The following code demonstrates the use of the `using` keyword with namespaces:

Snippet

```
using System;
class World
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello World");
    }
}
```

- In the code:
  - The `System` namespace is imported within the program with the `using` keyword.
  - If this were not done, the program would not even compile as the `Console` class exists in the `System` namespace.

Output

```
Hello World
```

Using the System Namespace 6-6

- The following code refers to the `Console` class of the `System` namespace multiple times:

Snippet

```
class World
{
   static void Main(string[] args)
   {
       System.Console.WriteLine("Hello World");
       System.Console.WriteLine("This is C# Programming");
       System.Console.WriteLine("You have executed a simple program of
       C#");
   }
}
```

- In the code, the class is not imported, but the `System` namespace members are used along with the statements.

Output

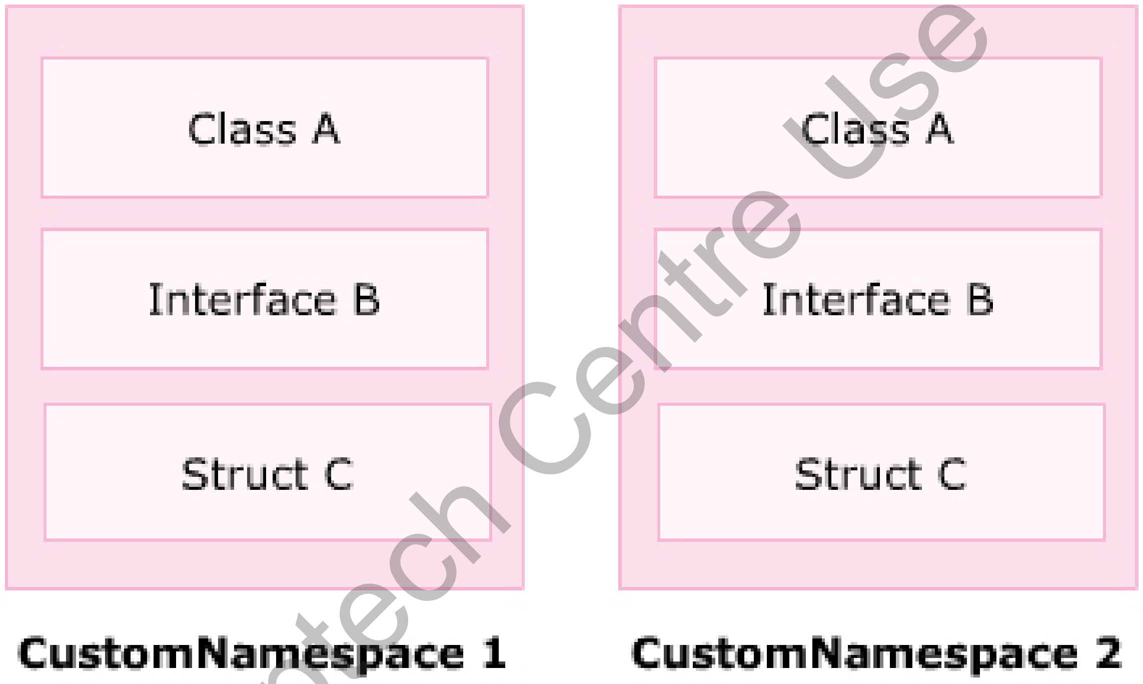
```
Hello World
This is C# Programming
You have executed a simple program of C#
```

Custom Namespaces 1-9

- C# allows you to create namespaces with appropriate names to organize structures, classes, interfaces, delegates, and enumerations that can be used across different C# applications.
- When using a custom namespace, you need not worry about name clashes with classes, interfaces, and so on in other namespaces.
- Custom namespaces:
    - Enable you to control the scope of a class by deciding the appropriate namespace for the class.
    - Declared using the `namespace` keyword and is accessed with the `using` keyword similar to any built-in namespace.

Custom Namespaces 2-9

- The following figure displays a general example of using custom namespaces:



Custom Namespaces 3-9

- The following syntax is used to declare a custom namespace:

Syntax

```
namespace <NamespaceName>
{
//type-declarations;
}
```

- In the syntax:
  - `NamespaceName`: Is the name of the custom namespace.
  - `type-declarations`: Are the different types that can be declared. It can be a class, interface, struct, enum, delegate, or another namespace.

Custom Namespaces 4-9

- The following code creates a custom namespace named **`Department`**:

Snippet

```
namespace Department
{
    class Sales
    {
    static void Main(string [] args)
    {
        System.Console.WriteLine("You have created a custom namespace
        named Department");
    }
    }
}
```

- In the code:
  - **`Department`** is declared as the custom namespace.
  - The class **`Sales`** is declared within this namespace.

Output

```
You have created a custom namespace named Department
```

Custom Namespaces 5-9

- Once a namespace is created, C# allows:
  - Additional classes to be included later in that namespace. Hence, namespaces are additive.
  - A namespace to be declared more than once.
- These namespaces can be split and saved in separate files or in the same file.
- At the time of compilation, these namespaces are added together.

Custom Namespaces 6-9

- The namespace split over multiple files is illustrated in the next three code snippets:

Snippet

```
// The Automotive namespace contains the class SpareParts and
// this namespace is partly stored in the SpareParts.cs file.

using System;
namespace Automotive
{
   public class SpareParts
   {
      string _spareName;
      public SpareParts()
      {
         _spareName = "Gear Box";
      }
      public void Display()
      {
         Console.WriteLine("Spare Part name: " + _spareName);
      }
   }
}
```

Custom Namespaces 7-9

Snippet

```
//The Automotive namespace contains the class Category and
// this namespace is partly stored in the Category.cs file.

using System;
namespace Automotive
{
   public class Category
   {
      string _category;
      public Category()
      {
        _category = "Multi Utility Vehicle";
      }
      public void Display()
      {
        Console.WriteLine("Category: " + _category);
      }
   }
}
```

Custom Namespaces 8-9

Snippet

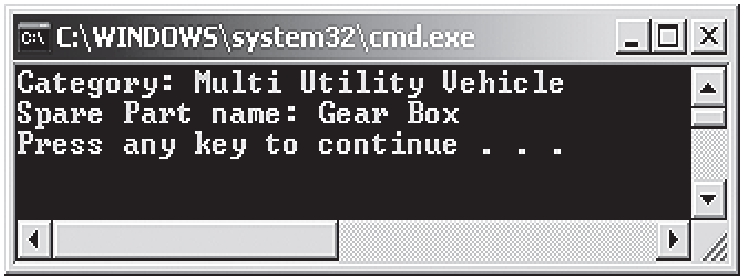
```
//The Automotive namespace contains the class Toyota and this
// namespace is partly stored in the Toyota.cs file.

namespace Automotive
{
   class Toyota
   {
    static void Main(string[] args)
    {
       Category objCategory = new Category();
       SpareParts objSpare = new SpareParts();
       objCategory.Display();
       objSpare.Display();
     }
   }
}
```

Custom Namespaces 9-9

- In the three code snippets:
  - The three classes, **`SpareParts`**, **`Category`**, and **`Toyota`** are stored in three different files, **`SpareParts.cs`**, **`Category.cs`**, and **`Toyota.cs`** respectively.
  - Even though the classes are stored in different files, they are still in the same namespace, namely **`Automotive`**. Hence, a reference is not required here.
  - A single namespace **`Automotive`** is used to enclose three different classes.
  - The code for each class is saved as a separate file.
  - When the three files are compiled, the resultant namespace is still **`Automotive`**.
- The following figure shows the output of the application containing the three files:

Output



```
Category: Multi Utility Vehicle
Spare Part name: Gear Box
Press any key to continue . . .
```

Guidelines for Creating Custom Namespaces 1-2

- While designing a large framework for a project, it is required to create namespaces to group the types into the appropriate namespaces such that the identical types do not collide.

- Therefore, the following guidelines must be considered for creating custom namespaces:
  - All similar elements such as classes and interfaces must be created into a single namespace. This will form a logical grouping of similar types and any programmer will be easily able to search for similar classes.
  - Creating deep hierarchies that are difficult to browse must be avoided.
  - Creating too many namespaces must be avoided for simplicity.

Guidelines for Creating Custom Namespaces 2-2

- Nested namespaces must contain types that depend on the namespace within which it is declared.

Example

- The classes in the **`Country.States.Cities`** namespace will depend on the classes in the namespace **`Country.States`**.

Example

- If a user has created a class called **`US`** in the **`States`** namespace, only the cities residing in U.S. can appear as classes in the **`Cities`** namespace.

Access Modifiers for Namespaces 1-2

- Namespaces are always implicitly public.
- You cannot apply access modifiers such as `public`, `protected`, `private`, or `internal` to namespaces.
- The namespace is accessible by any other namespace or class that exists outside the accessed namespace.
- The access scope of a namespace cannot be restricted.

Access Modifiers for Namespaces 2-2

- If any of the access modifiers is specified in the namespace declaration, the compiler generates an error.
- The following code attempts to declares the namespace as `public`:

Snippet

```
using System;
public namespace Products
{
   class Computers
   {
        static void Main(string [] args)
        {
            Console.WriteLine("This class provides information
            about Computers");
        }
   }

}
```

- The code generates the error, `'A namespace declaration cannot have modifiers or attributes'`.

Unqualified Naming 1-2

- A class defined within a namespace is accessed only by its name.
- To access this class, you just have to specify the name of that class.
- This form of specifying a class is known as Unqualified naming.
- The use of unqualified naming results in short names and can be implemented by the `using` keyword.
- This makes the program simple and readable.

Unqualified Naming 2-2

- The following code displays the student's name, ID, subject, and marks scored using an unqualified name:

Snippet

```
using System;
using Students;
namespace Students
{
     class StudentDetails
     {
          string _studName = "Alexander";
          int _studID = 30;
          public StudentDetails()
          {
               Console.WriteLine("Student Name: " +
               _studName);
               Console.WriteLine("Student ID: " + _studID);
          }
     }

}
namespace Examination
{
     class ScoreReport
     {
          public string Subject = "Science";
          public int Marks = 60;
          static void Main(string[] args)
          {
               StudentDetails objStudents = new
               StudentDetails();
               ScoreReport objReport = new ScoreReport();
               Console.WriteLine("Subject: " +
               objReport.Subject);
               Console.WriteLine("Marks: " + objReport.Marks);
          }
     }
}
```

- In the code, the class **`ScoreReport`** uses the class **`StudentDetails`** defined in the namespace `Examination`. The class is accessed by its name.

Qualified Naming 1-2

- C# allows you to use a class outside its namespace.
- A class outside its namespace can be accessed by specifying its namespace followed by the dot operator and the class name.
- This form of specifying the class is known as **Fully Qualified naming**.
- The use of fully qualified names results in long names and repetition throughout the code.
- Instead, you can access classes outside their namespaces with the `using` keyword.
- This makes the names short and meaningful.

Qualified Naming 2-2

- The following code displays the student’s name, ID, subject and marks scored using a fully qualified name:

Snippet

```
using System;
namespace Students
{
      class StudentDetails
      {
          string _studName = “Alexander”;
          int _studId = 30;
          public StudentDetails()
          {
              Console.WriteLine(“Student Name: “ + _studName);
              Console.WriteLine(“Student ID: “ + _studId);
          }
      }
}
namespace Examination
{
      class ScoreReport
      {
          public string Subject = “Science”;
          public int Marks = 60;
          static void Main(string[] args)
          {
               Students.StudentDetails objStudents = new Students.
               StudentDetails();
               ScoreReport objReport = new ScoreReport();
               Console.WriteLine(“Subject: “ + objReport.Subject);
               Console.WriteLine(“Marks: “ + objReport.Marks);
          }
      }
}
```

- In the code, the class **`ScoreReport`** uses the class **`StudentDetails`** defined in the namespace **`Examination`**. The class is accessed by its fully qualified name.

Naming Conventions for Namespaces

- When namespaces are being created to handle projects of an organization, it is recommended that namespaces are prefixed with the company name followed by the technology name, the feature, and the design of the brand.
- Namespaces for projects of an organization can be created as follows:

```
CompanyName.TechnologyName[.Feature][.Design]
```

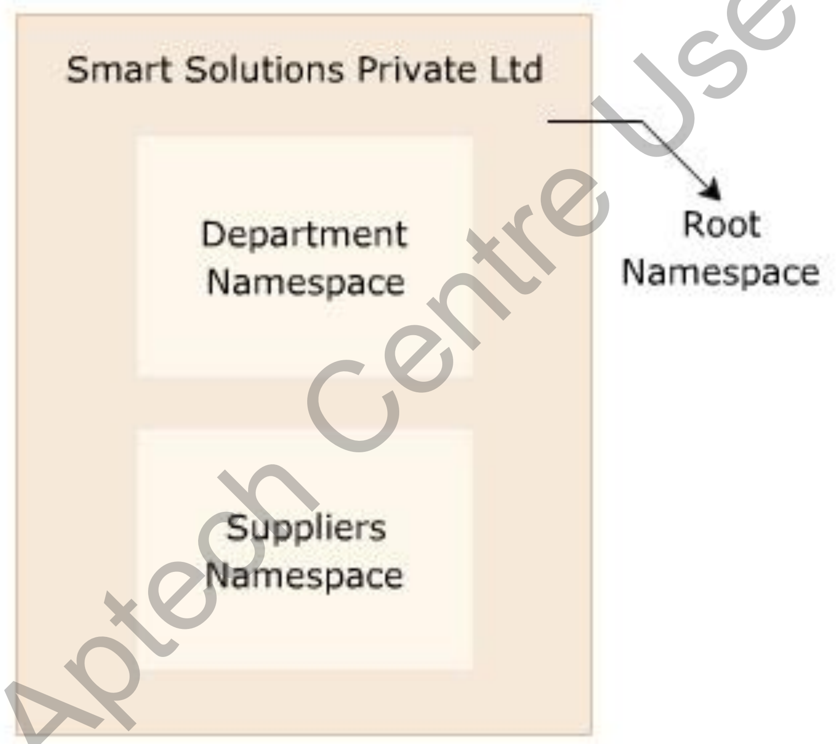
- Naming conventions that should be followed for creating namespaces are:
  - Use Pascal case for naming the namespaces.
  - Use periods to separate the logical components.
  - Use plural names for namespaces wherever applicable.
  - Ensure that a namespace and a class do not have same names.
  - Ensure that the name of a namespace is not identical to the name of the assembly.

Nested Namespaces 1-2

- C# allows you to define namespaces within a namespace.
- This arrangement of namespaces is referred to as nested namespaces.
- For an organization running multiple projects, nested namespaces is useful.
- The root namespace can be given the name of the organization and nested namespaces can be given the names of individual projects or modules.
- This allows the developers to store commonly used classes in appropriate namespaces and use them for all other similar programs.

Nested Namespaces 2-2

- The following figure illustrates the concept of nested namespaces:



Nested Namespaces for a Company

Implementing Nested Namespaces 1-3

- C# allows you to create a hierarchy of namespaces by creating namespaces within namespaces.
- Such nesting of namespaces is done by enclosing one namespace declaration inside the declaration of the other namespace.
- The following syntax can be used to create nested namespaces:

Syntax

```
namespace <NamespaceName>
{
   namespace <NamespaceName>
   {
   }
   namespace <NamespaceName>
   {
   }
}
```

Implementing Nested Namespaces 2-3

- The following code creates nested namespaces:

Snippet

```
namespace Contact
{
    public class Employees
    {
       public int EmpID;
    }
    namespace Salary
    {
       public class SalaryDetails
       {
           public double EmpSalary;
       }
    }

}
```

- In the code:
  - **`Contact`** is declared as a custom namespace that contains the class **`Employees`** and another namespace **`Salary`**.
  - The **`Salary`** namespace contains the class **`SalaryDetails`**.

Implementing Nested Namespaces 3-3

- The following code displays the salary of an employee using the nested namespace that was created in the previous code:

Snippet

```
using System;
class EmployeeDetails
{
     static void Main(string [] args)
     {
          Contact.Salary.SalaryDetails objSal = new
          Contact.Salary.SalaryDetails();
          objSal.EmpSalary = 1000.50;
          Console.WriteLine("Salary: " + objSal.EmpSalary);
     }
}
```

Output

```
Salary: 1000.5
```

- In the **`EmployeeDetails`** class of the code:
  - The object of the **`SalaryDetails`** class is created using the namespaces in which the classes are declared.
  - The value of the variable **`EmpSalary`** of the **`SalaryDetails`** class is initialized to `1000.5` and the salary amount is displayed as the output.

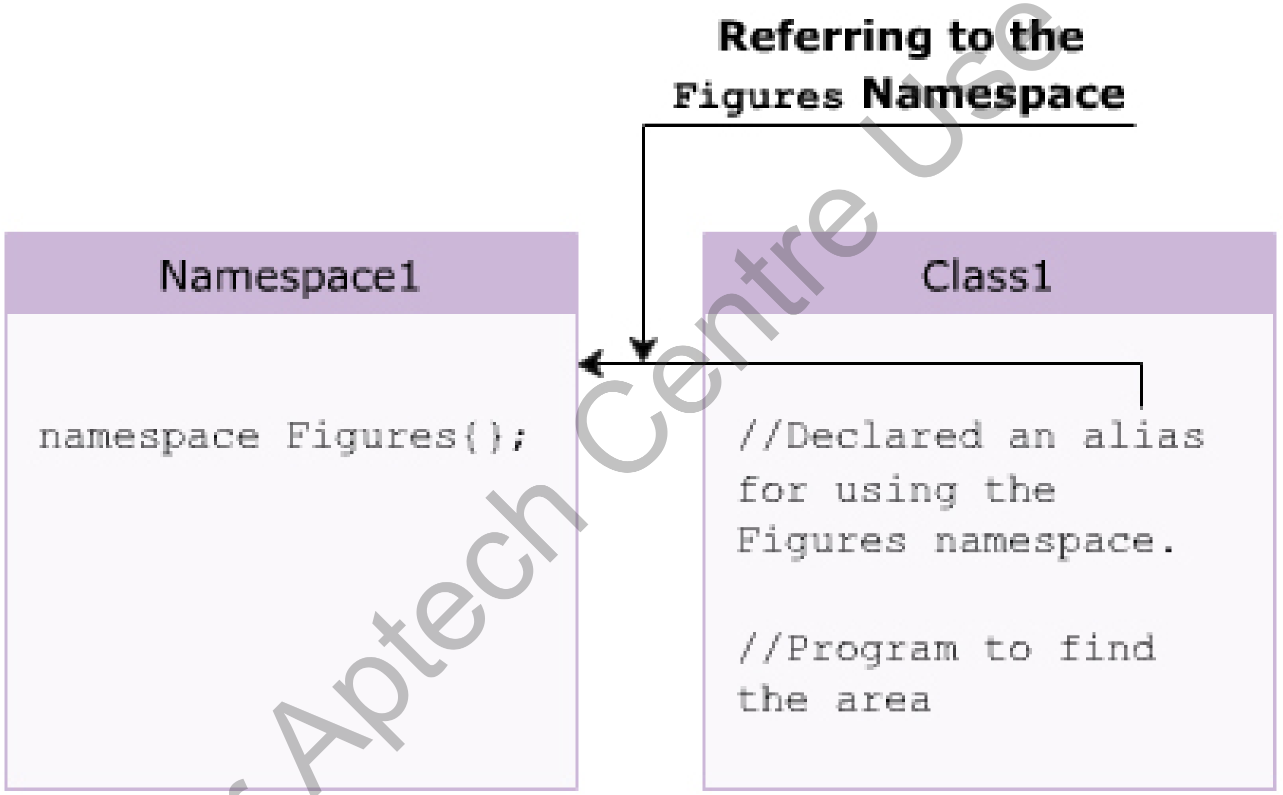
Namespace Aliases 1-5

- Aliases:
  - Are temporary alternate names that refer to the same entity.
  - Are also useful when a program contains many nested namespace declarations and you would like to distinguish as to which class belongs to which namespace.
  - Would make the code more readable for other programmers and would make it easier to maintain.

- The namespace referred to with the `using` keyword refers to all the classes within the namespace.
- However, sometimes you might want to access only one class from a particular namespace.
- You can use an alias name to refer to the required class and to prevent the use of fully qualified names.

Namespace Aliases 2-5

- The following figure displays an example of using namespace aliases:


Referring to the Figures Namespace
Namespace1
namespace Figures();
Class1
//Declared an alias for using the Figures namespace.
//Program to find the area

Namespace Aliases 3-5

- The following syntax is used for creating a namespace alias:

Syntax

```
using<aliasName> = <NamespaceName>;
```

- In the code:
  - `aliasName`: Is the user-defined name assigned to the namespace.

Namespace Aliases 4-5

- The following creates a custom namespace called **`Bank.Accounts.EmployeeDetails`**:

Snippet

```
namespace Bank.Accounts.EmployeeDetails
{
     public class Employees
     {
        public string EmpName;
     }
}
```

- The following code displays the name of an employee using the aliases of the `System.Console` and **`Bank.Accounts.EmployeeDetails`** namespaces:

Snippet

```
using IO = System.Console;
using Emp = Bank.Accounts.EmployeeDetails;
class AliasExample
{
    static void Main (string[] args)
    {
         Emp.Employees objEmp = new Emp.Employees();
         objEmp.EmpName = "Peter";
         IO.WriteLine("Employee Name: " + objEmp.EmpName);
    }
}
```

Namespace Aliases 5-5

Output

```
Employee Name: Peter
```

- In the code:
  - The **`Bank.Accounts.EmployeeDetails`** is aliased as **`Emp`** and `System.Console` is aliased as **IO**.
  - These alias names are used in the **`AliasExample`** class to access the **`Employees`** and **`Console`** classes defined in the respective namespaces.

External Aliasing 1-3

Example

- Consider a large organization working on multiple projects.
- The programmers working on two different projects may use the same class name belonging to the same namespace and store them in different assemblies.
- The assemblies created can also contain similar method names.
- When these assemblies are used in a single program and a common method from these assembles is invoked, compilation error is generated.
- The compilation error occurs since the same method resides in both the assemblies.
- This is called an ambiguous reference and it can be resolved by implementing external aliasing.

External Aliasing 2-3

- External aliasing in C#:
  - Allows the users to define assembly qualified namespace aliases.
  - Can be implemented using the `extern` keyword.
- This is illustrated in the following code:

Snippet

```
extern alias LibraryOne;
extern alias LibraryTwo;
using System;
class Companies
{
   static void Main (string [] args)
   {
      LibraryOne::Employees.Display();
      LibraryTwo::Employees.Display();
   }
}
```

External Aliasing 3-3

- In the code:
    - The **`Companies`** class references to two different assemblies in which the **`Employees`** class is created with the **`Display()`** method.
    - The external aliases for the two assemblies are defined as **`LibraryOne`** and **`LibraryTwo`**.
    - During the compilation of this code, the aliases must be mapped to the path of the respective assemblies through the compiler options.
    - For example, you may write:

```
/reference: LibraryOne =One.dll
/reference: LibraryTwo =two.dll
```

Namespace Alias Qualifier 1-10

- There are some situations wherein the alias provided to a namespace matches with the name of an existing namespace.
- Then, the compiler generates an error while executing the program that references to that namespace.
- This is illustrated in the following code:

Snippet

```
//The  following  program  is  saved  in  the  Automobile.cs  file
// under the Automotive project.

    using System;
    using System.Collections.Generic;
    using System.Text;
    using Utility_Vehicle.Car;
    using Utility_Vehicle = Automotive.Vehicle.Jeep;

    namespace Automotive
```

Namespace Alias Qualifier 2-10

```
    {
        namespace Vehicle
        {
            namespace Jeep
            {
                class Category
                {
                 string _category;
                 public Category()
                 {
                 _category = "Multi Utility Vehicle";
                 }
                 public void Display()
                 {
                 Console.WriteLine("Jeep Category: " +
                 _category);
                 }
                }
            }
          class Automobile
          {
            static void Main(string[] args)
            {
                Category objCat = new Category();
                objCat.Display();
                Utility_Vehicle.Category objCategory = new
                Utility_Vehicle.Category();
                objCategory.Display();
            }
        }
    }
}
```

Namespace Alias Qualifier 3-10

- Another project is created under the **Automotive** project to store the program shown in the following code:

Snippet

```
//The following program is saved in the Category.cs file under the
//Utility_Vehicle project created under the Automotive project.
using System;
using System.Collections.Generic;
using System.Text;

namespace Utility_Vehicle
{
      namespace Car
      {
           class Category
           {
                string _category;
                public Category()
                {
                   _category = "Luxury Vehicle";
                }
                public void Display()
                {
                    Console.WriteLine("Car Category: " +
                    _category);
                }
            }
       }
}
```

Namespace Alias Qualifier 4-10

- Both these files are compiled using the following syntax:

Syntax

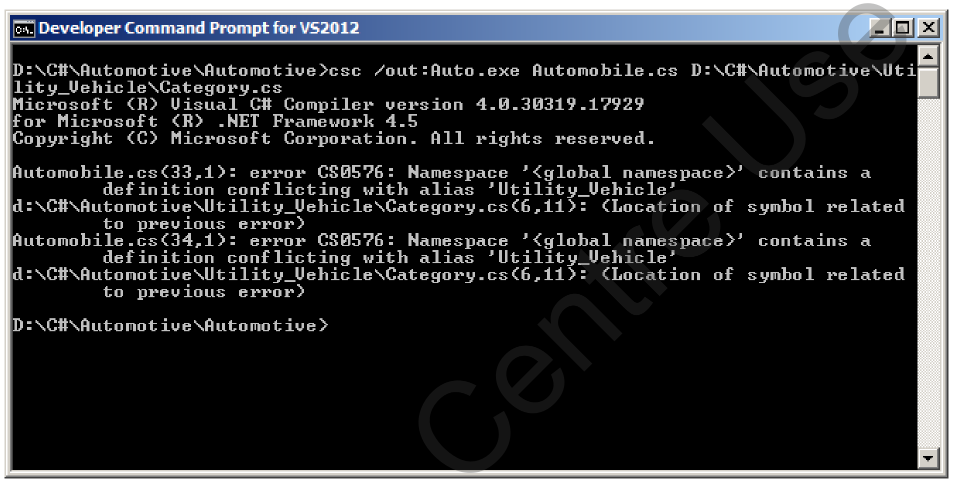
```
csc /out:<FileName>.exe <FullPath of File1>.cs <Full Path
   of File2>.cs
```

- In the syntax:
    - `FileName`: Is the name of the single `.exe` file that will be generated.
    - `FullPath of File 1`: Is the complete path where the first file is located.
    - `FullPath of File 2`: Is the complete path where the second file is located.

Namespace Alias Qualifier 5-10

- The following figure shows the outcome of the compiled code:

Output



Developer Command Prompt for VS2012

```
D:\C#\Automotive\Automotive>csc /out:Auto.exe Automobile.cs D:\C#\Automotive\Uti
lity_Vehicle\Category.cs
Microsoft (R) Visual C# Compiler version 4.0.30319.17929
for Microsoft (R) .NET Framework 4.5
Copyright (C) Microsoft Corporation. All rights reserved.

Automobile.cs(33,1): error CS0576: Namespace '<global namespace>' contains a
        definition conflicting with alias 'Utility_Vehicle'
d:\C#\Automotive\Utility_Vehicle\Category.cs(6,11): (Location of symbol related
        to previous error)
Automobile.cs(34,1): error CS0576: Namespace '<global namespace>' contains a
        definition conflicting with alias 'Utility_Vehicle'
d:\C#\Automotive\Utility_Vehicle\Category.cs(6,11): (Location of symbol related
        to previous error)

D:\C#\Automotive\Automotive>
```

- In both the codes:
  - The namespaces **`Jeep`** and **`Car`** include the class **`Category`**.
  - An alias **`Utility_Vehicle`** is provided to the namespace **`Automotive.Jeep`**.
  - This alias matches with the name of the other namespace in which the namespace **`Car`** is nested.
  - To compile both the programs, the `csc` command is used which uses the complete path to refer to the **`Category.cs`** file.
  - In the **`Automobile.cs`** file, the alias name **`Utility_Vehicle`** is the same as the namespace which is being referred by the file. Due to this name conflict, the complier generates an error.

Namespace Alias Qualifier 6-10

- This problem of ambiguous name references can be resolved by using the namespace alias qualifier.
- Namespace alias qualifier is a new feature of C# and it can be used in the form of:

Syntax

```
<LeftOperand> :: <RightOperand>
```

- In the syntax:
  - `LeftOperand`: Is a namespace alias, an extern, or a global identifier.
  - `RightOperand`: Is the type.

Namespace Alias Qualifier 7-10

- The correct code for this is given as follows:

Snippet

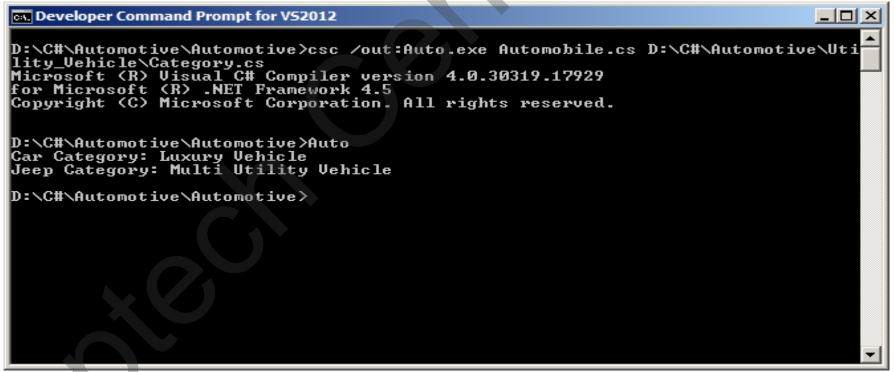
```
using System;
using System.Collections.Generic;
using System.Text;
using Utility_Vehicle.Car;
using Utility_Vehicle = Automotive.Vehicle.Jeep;

namespace Automotive
{
     namespace Vehicle
     {
       namespace Jeep
       {
         class Category
         {
          string _category;
          public Category()
          {
          _category = "Multi Utility Vehicle";
          }
          public void Display()
          {
          Console.WriteLine("Jeep Category: " +
          _category);
          }
        }
     }
```

Namespace Alias Qualifier 8-10

```
        class Automobile
        {
          static void Main(string[] args)
          {
              Category objCat = new Category();
              objCat.Display();
              Utility_Vehicle::Category objCategory = new
              Utility_Vehicle::Category();
              objCategory.Display();
          }
        }
    }
}
```

Output



```
D:\C#\Automotive\Automotive>csc /out:Auto.exe Automobile.cs D:\C#\Automotive\Uti
lity_Vehicle\Category.cs
Microsoft (R) Visual C# Compiler version 4.0.30319.17929
for Microsoft (R) .NET Framework 4.5
Copyright (C) Microsoft Corporation. All rights reserved.


D:\C#\Automotive\Automotive>Auto
Car Category: Luxury Vehicle
Jeep Category: Multi Utility Vehicle

D:\C#\Automotive\Automotive>
```

- In the code:
  - In the class **`Automobile`**, the namespace alias qualifier is used.
  - The alias **`Utility_ Vehicle`** is specified in the left operand and the class **`Category`** in the right operand.

Namespace Alias Qualifier 9-10

- The alias qualifier is used when a user needs to define the class `System` in the custom namespace.
- C# does not allow users to create a class named `System`.
- If a class named `System` is created and a constant variable named `Console` is declared within this class, the compiler will generate an error on using the `System.Console` class.
- This is because C# includes the system defined namespace called `System`.
- This problem can be resolved by using namespace alias qualifier with its left operand defined as global as shown in the following code:

Snippet

```
using System;
namespace ITCompany
{
   class System
```

Namespace Alias Qualifier 10-10

```
        {
        const string Console = "Console";
        public static string WriteLine()
        {
          return "WriteLine method of my System class";
        }
        static void Main(string[] args)
        {
          global::System.Console.WriteLine (WriteLine());
        }
    }
}
```

Output

```
WriteLine method of my System class
```

- In the code:
  - The namespace alias qualifier is used in the class **`System`**.
  - The left operand of the namespace alias qualifier is specified as global using the `global` keyword.
  - Therefore, the search for the right operand starts at the global namespace.

Summary

- A namespace in C# is used to group classes logically and prevent name clashes between classes with identical names.
- The System namespace is imported by default in the .NET Framework.
- Custom namespaces enable you to control the scope of a class by deciding the appropriate namespace for the class.
- You cannot apply access modifiers such as public, protected, private, or internal to namespaces.
- A class outside its namespace can be accessed by specifying its namespace followed by the dot operator and the class name.
- C# supports nested namespaces that allows you to define namespaces within a namespace.
- External aliasing in C# allows the users to define assembly qualified namespace aliases.