## **Nested Types**

A **nested type** is a type (such as a class, struct, interface, enum, or delegate) that is declared inside the scope of another class or struct, which is called the **enclosing type**.Tipe bersarang adalah tipe (seperti kelas, struktur, antarmuka, enum, atau delegasi) yang dideklarasikan di dalam ruang lingkup kelas atau struktur lain, yang disebut tipe penutup.

Here's a simple illustration:

|  |
| --- |
| public class TopLevel // Enclosing class {  public class Nested { } // Nested class  public enum Color { Red, Blue, Tan } // Nested enum } |

### **Key Features of Nested Types:**

1. **Access to Enclosing Type's Members:** A nested type has special access privileges. It can directly access **all members** of its enclosing type, including private members. This is a crucial advantage when the nested type needs to intimately interact with the internals of its container.Akses ke Anggota Tipe Penutup: Tipe bersarang memiliki hak akses khusus. Ia dapat mengakses langsung semua anggota tipe penutupnya, termasuk anggota privat. Keuntungan ini sangat penting ketika tipe bersarang perlu berinteraksi secara mendalam dengan bagian dalam kontainernya.

|  |
| --- |
| public class TopLevel {  static int x = 10; // Private static field   class Nested // Nested class  {  static void Foo()  {  Console.WriteLine(TopLevel.x); // Nested class can access private static member 'x'  }  } } |

1. **Full Range of Access Modifiers:** Unlike non-nested types (which default to internal and are commonly public), nested types can be declared with the full range of access modifiers (public, internal, private, protected, protected internal, private protected, file). This offers fine-grained control over their visibility.Rentang Penuh Modifier Akses: Berbeda dengan tipe non-bersarang (yang secara default memiliki akses internal dan umumnya publik), tipe bersarang dapat dideklarasikan dengan rentang penuh modifier akses (publik, internal, privat, terlindungi, terlindungi internal, privat terlindungi, file). Hal ini memberikan kontrol yang lebih detail atas visibilitasnya.

|  |
| --- |
| public class TopLevel {  protected class Nested { } // Nested class can be protected }  public class SubTopLevel : TopLevel {  static void Foo()  {  new TopLevel.Nested(); // OK: SubTopLevel can access protected Nested  } } |

1. **Default Accessibility is private:** For members of a class or struct, the default accessibility is private. This applies to nested types as well. If you don't specify an access modifier for a nested type, it will be private.Aksesibilitas default adalah private: Untuk anggota kelas atau struktur, aksesibilitas default adalah private. Hal ini juga berlaku untuk tipe yang tertanam. Jika Anda tidak menentukan modifikator akses untuk tipe yang tertanam, maka tipe tersebut akan menjadi private.

|  |
| --- |
| public class OuterClass {  class InnerClass { } // InnerClass is private by default } // Cannot access OuterClass.InnerClass from outside OuterClass. |

1. This differs from non-nested types, which default to internal.Hal ini berbeda dengan tipe non-nested, yang secara default menggunakan akses internal.
2. **Qualification for External Access:** To access a nested type from outside its enclosing type, you must **qualify** its name with the enclosing type's name, similar to how you access static members.Persyaratan Akses Eksternal: Untuk mengakses tipe bersarang dari luar tipe yang mengelilinginya, Anda harus menyertakan nama tipe yang mengelilinginya di depan nama tipe bersarang tersebut, mirip dengan cara mengakses anggota statis.

|  |
| --- |
| public class TopLevel {  public class Nested { }  public enum Color { Red, Blue, Tan } }  class Test {  TopLevel.Nested n; // Qualified access for nested class  TopLevel.Color color = TopLevel.Color.Red; // Qualified access for nested enum } |

### **Types That Can Be Nested:**

All types in C# can be nested:

* Classes within classes or structs.
* Structs within classes or structs.
* Interfaces within classes or structs.
* Delegates within classes or structs.**tipe data yang mereferensikan sebuah metode**
* Enums within classes or structs.

### **When to Use Nested Types:**

Nested types are a useful organizational tool. They are particularly appropriate in scenarios where:Jenis bertingkat adalah alat organisasi yang berguna. Mereka sangat cocok dalam skenario di mana:

* **Stronger Access Control:** You need tight access control over a helper type that is conceptually part of another type's implementation and should not be exposed globally.Kontrol Akses yang Lebih Ketat: Anda memerlukan kontrol akses yang ketat terhadap jenis bantu yang secara konseptual merupakan bagian dari implementasi jenis lain dan tidak boleh diakses secara global.
* **Intimate Collaboration:** The nested type needs to access private members of the enclosing type to perform its function. This eliminates the need for exposing internal state just for the helper type.Kerja Sama yang Erat: Tipe bersarang memerlukan akses ke anggota privat dari tipe yang mengelilinginya untuk menjalankan fungsinya. Hal ini menghilangkan kebutuhan untuk mengekspos keadaan internal hanya untuk tipe bantu.
* **Logical Grouping/Readability:** The nested type is conceptually related to and primarily used by its enclosing type, making the code more organized and easier to understand.Pengelompokan Logis/Kemudahan Membaca: Tipe bersarang secara konseptual terkait dengan dan terutama digunakan oleh tipe yang mengelilinginya, sehingga kode menjadi lebih terorganisir dan lebih mudah dipahami.
* **Avoiding Namespace Clutter:** While a nested namespace can also help organize types, a nested type goes further by providing stronger encapsulation and direct access to the enclosing type's private members.Menghindari Kerumitan Namespace: Meskipun namespace bersarang juga dapat membantu mengorganisir tipe, tipe bersarang memberikan enkapsulasi yang lebih kuat dan akses langsung ke anggota privat tipe yang mengelilinginya.

The C# compiler itself heavily uses nested types when it generates private classes to manage the state for advanced constructs like iterators (yield return) and anonymous methods (lambda expressions). This demonstrates their utility for internal, highly coupled implementations.Penerjemah C# sendiri secara intensif menggunakan tipe bersarang saat menghasilkan kelas privat untuk mengelola keadaan pada konstruksi lanjutan seperti iterator (yield return) dan metode anonim (ekspresi lambda). Hal ini menunjukkan kegunaannya untuk implementasi internal yang sangat terintegrasi.