Var, Enum, public, private and protected

c#

var Keyword in C#

The `var` keyword is used to declare variables implicitly, allowing the compiler to determine the variable's type at compile time.

Usage:

- **Example:**

```
var name = "John";     // inferred as string

var age = 25;              // inferred as int

var isActive = true;  // inferred as bool
```

Key Points:

- `var` can only be used within local scope (method, loop, etc.).
- The type must be determinable at compile time.
- Improves readability when the type is obvious, but overuse can reduce code clarity.

What is an Enum in C#?

- An **enum** (short for "enumeration") is a special data type in C# that allows you to define a set of named **constants**. These constants are represented by integers under the hood, but they make your code more readable and easier to manage when dealing with fixed sets of values.

Why Use Enums?

Enums are useful when you have a collection of related constants, such as days of the week, months of the year, or order statuses, and you want to represent them in a meaningful and readable way.

Simple Explanation:

- Instead of using raw numbers or strings to represent things, you can use enums to give them meaningful names.
- Enums are **type-safe**, meaning you can’t assign any other values (like random integers or strings) to them, which reduces errors.

Basic Example: Days of the Week

Imagine you want to work with days of the week in your program. Without an enum, you might use integers or strings:

- **Without Enum:**

```
int day = 1; // Is 1 Monday or Sunday?

string day = "Monday"; // What if you misspell it as "Monday"?
```

This can lead to confusion and errors. Instead, you can use an enum to represent the days clearly:

Defining an Enum:

```
enum DaysOfWeek
{
    Sunday,    // 0
    Monday,    // 1
    Tuesday,   // 2
    Wednesday, // 3
    Thursday,  // 4
    Friday,    // 5
    Saturday   // 6
}
```

By default, the first value starts at `0`, and the next ones increase by `1`.

You can use these names (`Sunday`, `Monday`, etc.) in your code, which makes it easier to read and less prone to mistakes.

Using an Enum in Code:

```
class Program
{       static void Main()
        {
        DaysOfWeek today = DaysOfWeek.Monday;
        if (today == DaysOfWeek.Monday)
        {
        Console.WriteLine("Start of the workweek!");
        }
        // You can also print the value
        Console.WriteLine("Today is: " + today); // Output: Today is: Monday
        }
}
```

Explanation:

- Here, `today` is a variable of type `DaysOfWeek`.
- You can check the value of `today` using an if-statement, which is easier to understand than using integers.

Enum Behind the Scenes:

- Even though you use names like `Monday` or `Friday`, C# internally represents them as numbers (starting from `0` unless specified otherwise).

```
Console.WriteLine((int)DaysOfWeek.Monday); // Output: 1

Console.WriteLine((int)DaysOfWeek.Wednesday); // Output: 3
```

Custom Values in Enums:

You can assign custom values to the enum members if you don't want them to start at `0` or want specific numbers.

```
enum OrderStatus
{
    Pending = 1,
    Shipped = 5,
    Delivered = 10,
    Cancelled = 15
}
```

Here, each status has a custom integer value. You can use it like this:

```
class Program
{
    static void Main()
    {
    OrderStatus status = OrderStatus.Shipped;
    Console.WriteLine("Order status is: " + status); // Output: Order status is: Shipped
    Console.WriteLine("Order status code is: " + (int)status); // Output: Order status code is: 5
    }
}
```

Advantages of Enums:

1. **Readability**: Instead of using numbers or strings, you use meaningful names.
2. **Type-Safety**: C# won't let you assign invalid values to an enum variable.
3. **Maintainability**: It's easier to add or update enum values without affecting the rest of the code.

In C#, the `base` keyword is used to refer to the parent class (also known as the base class) from within a derived class. It can be used to call a constructor of the parent class from the child class to ensure that the parent's constructor logic is executed.

Here's an example demonstrating how to use `base` to call the constructor of a parent class:

Example of Base Class Constructor Call

```
class Vehicle{

        public string Brand { get; private set; }

        public int Year { get; private set; }

        // Parent class constructor

        public Vehicle(string brand, int year)

        {

        Brand = brand;

        Year = year;

        Console.WriteLine($"Vehicle constructor called. Brand: {Brand}, Year: {Year}");

        }

}
```

```
class Car : Vehicle
{
        public string Model { get; private set; }

        // Child class constructor calling the parent class constructor
        public Car(string brand, int year, string model) : base(brand, year)
        {
        Model = model;
        Console.WriteLine($"Car constructor called. Model: {Model}");
        }
}
```

```
class Program
{
        static void Main(string[] args)
        {
        // Create an object of the Car class
        Car myCar = new Car("Toyota", 2020, "Corolla");


        // Output:
        // Vehicle constructor called. Brand: Toyota, Year: 2020
        // Car constructor called. Model: Corolla
        }
}
```

Access Modifiers in C#

Public, Private, and Protected:

- **public**: The member is accessible from any other class.
- **private**: The member is only accessible within the same class.
- **protected**: The member is accessible within the same class and in derived classes.

Comparison:

Modifier	Accessibility
`public`	Anywhere within the program
`private`	Only within the declaring class
`protected`	Declaring class and derived classes

Example:

```
public class Person
{
    public string Name { get; set; }   // Accessible everywhere
    private int age;                   // Accessible only within Person class
    protected string Gender { get; set; } // Accessible in derived classes
}
```

Public Modifier in C#

Definition:

- Members declared as `public` can be accessed from any other class.

Usage:

- **Example:**

```
public class Car
{
    public string Make { get; set; }        // Accessible everywhere
}
Car car = new Car();
car.Make = "Toyota";
```

Key Points:

- Used when you want the member to be globally accessible.

Private Modifier in C#

Definition:

- Members declared as `private` are only accessible within the same class.

Usage:

- **Example:**

```
class Account

{

        private decimal balance;


        public void Deposit(decimal amount)

        {

        balance += amount;  // Can modify balance within the same class

        }

}
```

Key Points:

- Provides encapsulation and security by restricting access to the member.

Protected Modifier in C#

Definition:

- Members declared as `protected` are accessible within their own class and by derived classes.

Usage:

- **Example:**

```
class Animal
{
        protected void Eat() { }
}

class Dog : Animal
{
        public void DogEat()
        {
        Eat();  // Accessible because Dog inherits Animal
        }
}
```