Object-Oriented Programming (OOP) in C#

By

Narasimha Rao T

Microsoft.Net FSD Trainer

Professional Development Trainer

tnrao.trainer@gmail.com

Index

1. Classes and Objects, Fields, Properties
2. Constructors (this keyword),
3. Encapsulation: access modifiers
4. Properties getters/setters
5. Static vs Instance

What is Object-Oriented Programming (OOP)?

Definition:

OOP is a programming paradigm based on the concept of **"objects"**, which contain both **data** and **behavior**.

Key Features:

- Promotes **modularity**, **reusability**, and **encapsulation**.
- Uses classes as blueprints for creating objects.

Advantages:

- Better code organization
- Easier maintenance
- Encourages reuse through inheritance and polymorphism

Introduction to OOP Principles – The 4 Pillars

1. Encapsulation

- Bundles data and methods into a single unit (class).
- Prevents unauthorized access using access modifiers.

```
class Person {
    private string name;
    public string Name {
        get { return name; }
        set { name = value; }
    }
}
```

2. Abstraction

- Hides internal implementation details.
- Focuses on what the object does instead of how.

```
abstract class Animal {
    public abstract void Speak();
}
```

3. Inheritance

- One class can inherit from another.
- Promotes code reuse.

```
class Animal {
    public void Eat() { }
}
class Dog : Animal {
    public void Bark() { }
}
```

4. Polymorphism

- One interface, multiple implementations.
- Achieved using method overriding or overloading.

```
class Animal {
    public virtual void Speak() {
        Console.WriteLine("Animal speaks");
    }
}
class Cat : Animal {
    public override void Speak() {
        Console.WriteLine("Cat meows");
    }
}
```

Classes in C#

Definition:

A class is a user-defined blueprint from which objects are created.

Structure:

```
class Car {
    public string Color;
    public void Drive() {
        Console.WriteLine("Car is driving.");
    }
}
```

Objects in C#

Definition:

An object is an instance of a class. It represents a real-world entity.

Creating an Object:

```
Car myCar = new Car();
myCar.Color = "Red";
myCar.Drive();
```

Constructors in C#

Definition:

A constructor is a special method that is called when an object is created.

Types:

- **Default constructor**
- **Parameterized constructor**

Example:

```
class Student {
    public string Name;

    public Student(string name) {
        Name = name;
    }
}
```

`this` Keyword

Definition:

`this` refers to the current instance of the class.

Use Cases:

- Resolve name conflicts between fields and parameters
- Pass the current instance as a parameter

Example

```
class Person {
    private string name;

    public Person(string name) {
        this.name = name; // distinguishes class field from parameter
    }
}
```

Properties in C#

Properties in C#

Definition:

Properties provide a flexible mechanism to read, write, or compute the values of private fields.

Types:

- **Standard Properties**
- **Auto-Implemented Properties**

Example:

```
class Person {
    private int age;

    public int Age {
        get { return age; }
        set {
            if (value >= 0)
                age = value;
        }
    }
}
```

Auto-Implemented Properties

Definition:

- Auto-implemented properties in C# provide a shorthand syntax for declaring properties **without explicitly defining a backing field**.
- The compiler automatically creates a private, anonymous field behind the scenes to store the property value.

Why Use Auto-Implemented Properties?

- Reduces boilerplate code.
- Useful when no custom logic is needed in the getter or setter.
- Keeps code cleaner and more readable.

Example

```
public string Name { get; set; }
```

The above is **equivalent to**:

```
private string _name;
public string Name {
    get { return _name; }
    set { _name = value; }
}
```

Static vs Instance

Q & A

Narasimha Rao T

tnrao.trainer@gmail.com