upGrad ENTERPRISE

Working with Delegates in C#

By
Narasimha Rao T
Microsoft.Net FSD Trainer
Professional Development Trainer
tnrao.trainer@gmail.com

1. Introduction to Delegates

Delegates in C# are **type-safe function pointers**. They allow you to **encapsulate method references**, meaning you can pass methods around as parameters, store them in variables, and invoke them dynamically.

Why Use Delegates?

- **Callbacks** (e.g., async operations)
- **Event handling** (e.g., UI button clicks)
- **Plug-in architectures**
- **Custom comparison and filtering**

2. What Are Delegates?

A delegate is a reference type that defines a method signature. Methods assigned to delegates must match this signature.

Declaration Syntax

```
public delegate int MathOperation(int a, int b);
```

Usage

```
public class Calculator
{
    public static int Add(int a, int b) => a + b;
    public static int Subtract(int a, int b) => a - b;
}

// Usage
MathOperation op = Calculator.Add;
Console.WriteLine(op(5, 3)); // Output: 8
```

3. Multicast Delegates

Multicast delegates can hold references to **multiple methods**.

```
public delegate void Notify();

public class Alerts
{
    public static void AlertA() => Console.WriteLine("Alert A triggered!");
    public static void AlertB() => Console.WriteLine("Alert B triggered!");
}

// Usage
Notify notify = Alerts.AlertA;
notify += Alerts.AlertB;

notify();
// Output:
// Alert A triggered!
// Alert B triggered!
```

4. Built-in Delegates: `Action`, `Func`, `Predicate`

C# provides **generic built-in delegate types**:

`Action` – No return value

```
Action<string> greet = name => Console.WriteLine($"Hello, {name}!");
greet("Alice");
```

`Func` – Returns a value

```
Func<int, int, int> multiply = (a, b) => a * b;
Console.WriteLine(multiply(3, 4)); // Output: 12
```

`Predicate` – Returns `bool`, typically used for filtering

```
Predicate<int> isEven = num => num % 2 == 0;
Console.WriteLine(isEven(10)); // True
```

5. Lambda Expressions

Lambda expressions are a concise way to write **anonymous methods**.

Basic Syntax

```
(x, y) => x + y
```

Expression-bodied Example

```
Func<int, int, int> add = (x, y) => x + y;
Console.WriteLine(add(5, 6)); // Output: 11
```

6. Passing Functions as Parameters

Delegates allow passing methods to other methods.

Example

```
public static void ExecuteOperation(int a, int b, Func<int, int, int> operation)
{
    Console.WriteLine($"Result: {operation(a, b)}");
}

// Usage
ExecuteOperation(5, 3, (x, y) => x * y); // Result: 15
```

7. Real-Time Examples

Game Engine (Event Handling)

```
public delegate void GameEvent();

public class Game
{
    public static void OnStart() => Console.WriteLine("Game started!");
    public static void OnEnd() => Console.WriteLine("Game over!");
}

// Usage
GameEvent gameEvents = Game.OnStart;
gameEvents += Game.OnEnd;

gameEvents();
```

E-commerce (Filtering Products)

```
public class Product
{
    public string Name { get; set; }
    public double Price { get; set; }
}

List<Product> products = new List<Product>
{
    new Product { Name = "Laptop", Price = 1500 },
    new Product { Name = "Mouse", Price = 25 },
};

Predicate<Product> isExpensive = p => p.Price > 100;
List<Product> expensiveItems = products.FindAll(isExpensive);
```

Calculator (Plug-in Operation)

```
public static double Operate(double x, double y, Func<double, double, double> operation)
{
    return operation(x, y);
}

double result = Operate(10, 5, (a, b) => a / b); // Division
```

Quiz Questions

1. What is a delegate in C#?
2. How does a delegate differ from a method?
3. What is the output of a multicast delegate if methods return values?
4. What is the difference between `Action` and `Func`?

upGrad ENTERPRISE

Q & A

Narasimha Rao T
tnrao.trainer@gmail.com