Classes

Published Oct 31, 2023

In C#, a **class** serves as a blueprint or template for creating objects. It plays a fundamental role in defining the structure, behavior, and attributes of these objects. A class can be thought of as a user-defined data type that encapsulates both data (attributes) and the actions (methods) that are applied to that data.

Syntax for Declaring Classes

To declare a class in C#, use the `class` keyword, followed by the class name. Class names should follow C# naming conventions (typically using PascalCase). The class definition is enclosed within curly braces `{}`.

```
public class MyClass {
    // Fields, properties, and methods go here
}
```

Properties and Methods

- `Properties`: Properties are used to define the attributes or data members of a class. They are defined within the class and provide access to the class's internal state.

- `Methods`: Methods are functions defined within the class that perform actions or operations. They can modify the class's state or provide functionality.

Access Modifiers

C# provides access modifiers to control the visibility and accessibility of class members. Common access modifiers include:

- `public`: Members are accessible from any code.
- `private`: Members are only accessible within the class.
- `protected`: Members are accessible within the class and derived classes.
- `internal`: Members are accessible within the same assembly (a group of related classes in the same project).
- `protected internal`: Members are accessible within the same assembly and derived classes.
- `private protected`: Members are accessible only from derived classes within the current assembly. This access modifier has been available since C# 7.2 and later.

Example

Here is a simple example featuring a class with properties and methods. By employing various access modifiers in C#, this example illustrates the use of public methods, such as `GetBalance`, to access private properties like the `current balance`. Meanwhile, it showcases that private methods like `PerformAudit()` are inaccessible from external code.

```
using System;

public class Program {
    public static void Main() {
        // Create a BankAccount instance with an initial balance of $1000.
        BankAccount account = new BankAccount(1000);

        // Use the GetBalance method to get the current balance.
        double currentBalance = account.GetBalance();
        Console.WriteLine("Current Balance: $" + currentBalance);

        // Attempt to call the private method PerformAudit (won't compile).
        // This will result in a compilation error because private methods can
        // account.PerformAudit();
```

```
    }
}

public class BankAccount {
    // Keeping track of current balance.
    private double balance;

    // Constructor
    public BankAccount(double initialAmount) {
        balance = initialAmount;
    }

    // Accessor method for balance.
    public double GetBalance() {
        return balance;
    }

    // Private method to perform a transaction audit.
    private void PerformAudit() {
        // In a real application, this method would perform auditing.
        // For this example, we'll just print a message.
        Console.WriteLine("Audit complete.");
    }
}
```

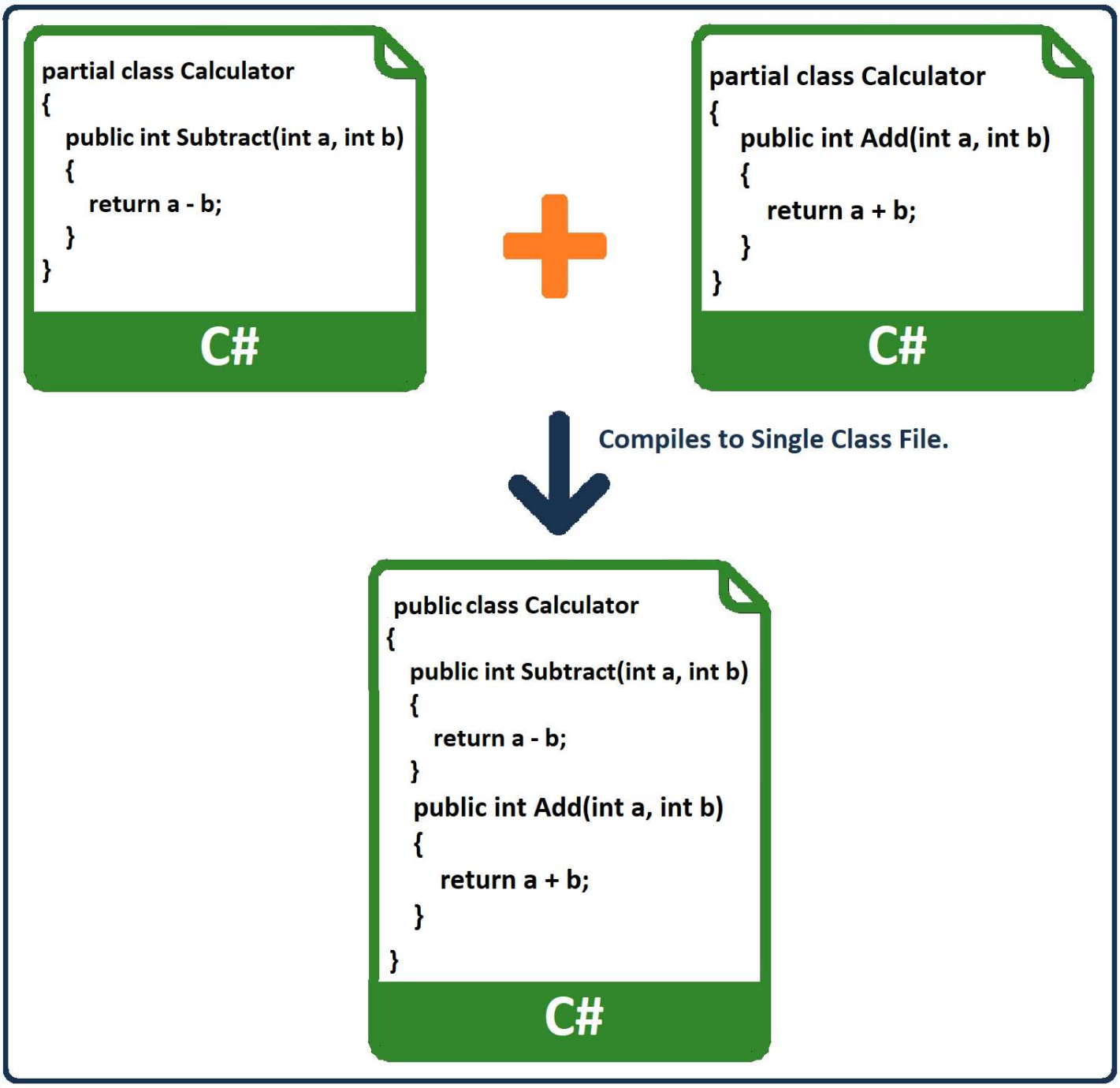
Static Classes

Static classes are defined using the `static` keyword and exclusively contain static members, such as methods, properties, and fields. Unlike regular classes, static classes cannot be instantiated with the `new` keyword. Instead, their members are accessed using the class name itself. These classes are commonly used for utility functions or to group related functionality.

Partial classes

Partial classes in C# enable class definitions to be split across multiple files. Each part of the class is defined in a separate file and combined at compile time to create a single class. This is

valuable for scenarios where a class becomes too large or complex, or when multiple developers need to work on different aspects of the class simultaneously.



In the image above, the `Calculator` class is depicted as a partial class structure, allowing independent development of class components. In the code below, the application's entry point, the `Main` method, creates an instance of the `Calculator` class and utilizes its methods for addition and subtraction operations.

```
using System;

class Program {
```

```
    static void Main() {
        Calculator calculator = new Calculator();
        int result1 = calculator.Add(5, 3);
        int result2 = calculator.Subtract(10, 4);

        Console.WriteLine("Addition: " + result1); // Output: Addition: 8
        Console.WriteLine("Subtraction: " + result2); // Output: Subtraction:
    }
}
```

All contributors

 Anonymous contributor

Contribute to Docs

- Learn more about how to get involved.
- Edit this page on GitHub to fix an error or make an improvement.
- Submit feedback to let us know how we can improve Docs.

Learn C# on Codecademy

Career path

Computer Science

Looking for an introduction to the theory behind programming? Master Python while learning data structures, algorithms, and more!

Includes **6 Courses**

With **Professional Certification**

Beginner Friendly **75** hours