Practical Application of Calling Methods

Introduction

Calling methods in C# allows you to reuse code, making your programs more efficient and easier to manage. This summary covers defining, calling, and using methods in C#, along with handling return values and passing arguments.

Basics of Calling Methods

A method is a reusable block of code designed to perform a task. It consists of a header, which defines its name, return type, and parameters, and a body containing the actual code. To call a method, use its name followed by parentheses, passing arguments if necessary:

```
int result = AddNumbers(5, 10);
```

This example calls the `AddNumbers` method, passing 5 and 10 as arguments and storing the result in the result.

Method Return Values

Methods often return values, defined by the return type in the header. If a method returns nothing, it uses void. For example, the `GetNumber` method below returns the integer 42:

```
public int GetNumber() {
  return 42;
}
```

The return statement sends this value back to the calling code.

Passing Arguments to Methods

Methods can accept input data through arguments. These must match the parameters defined in the method. For example, in the `CalculateSum` method, two integers are passed as arguments:

```
int sum = CalculateSum(5, 10);
```

This method returns the sum of the two numbers.

Practical Examples of Calling Methods

Displaying a Welcome Message

```
public void DisplayWelcomeMessage() {
  MessageBox.Show("Welcome to GreenWay!");
}
```

This method is called without arguments and displays a welcome message.

Calculating a Sum

```
public int CalculateSum(int a, int b) {
  return a + b;
}
```

The method adds two numbers and returns the result.

Validating User Age

```
public bool IsUserOldEnough(int age) {
  return age >= 18;
}
```

This method checks if a user is old enough to access certain features, returning true or false.

Conclusion

Mastering method calls in C# enables you to write modular, efficient code. Understanding how to define, pass arguments, and handle return values is essential for building maintainable and reusable programs.