3) 메서드와 구조체

매 강의 강의자료 시작에 PDF파일을 올려두었어요!

▼ PDF 파일

[수업 목표]

- C#에서 메서드와 구조체를 정의하고 사용하는 방법을 이해합니다.
- 메서드를 호출하고 반환값과 매개변수를 사용하는 방법을 익히고, 오버로딩을 이해합니다.
- 구조체를 생성하고 필드와 메서드를 사용하는 방법을 익힙니다.

[목차]

💡 모든 토글을 열고 닫는 단축키
Windows : `Ctrl` + `alt` + `t`
Mac : `⌘` + `⌥` + `t`

---

01. 메서드

✔️ 메서드를 활용해서 코드를 줄여봅시다

▼ 1) 메서드란?

- 메서드란?
  - 메서드(Method)는 일련의 코드 블록으로, 특정한 작업을 수행하기 위해 사용되는 독립적인 기능 단위입니다.
  - 코드의 재사용성과 모듈화를 위해 사용되며, 필요할 때 호출하여 실행됩니다.
- 메서드의 역할과 중요성
  - 코드의 재사용성:
    메서드를 사용하면 동일한 작업을 반복해서 구현하지 않아도 됩니다. 필요할 때 메서드를 호출하여 작업을 수행할 수 있습니다.
  - 모듈화:
    메서드를 사용하여 코드를 작은 단위로 분리하고 관리할 수 있습니다. 각 메서드는 특정한 기능을 수행하므로 코드의 구조가 더욱 명확해집니다.
  - 가독성과 유지보수성:
    메서드를 사용하면 코드가 간결해지고 가독성이 좋아집니다. 또한, 코드 수정이 필요한 경우 해당 메서드만 수정하면 되므로 유지보수가 용이해집니다.
  - 코드의 중복 제거:
    반복적인 작업을 메서드로 묶어서 사용하면 코드 중복을 방지할 수 있습니다.
  - 코드의 추상화:
    메서드를 통해 작업 단위를 추상화하고, 메서드 이름을 통해 해당 작업이 어떤 역할을 하는지 파악할 수 있습니다.

02. 메서드 선언과 호출

✔️ 메서드의 선언과 호출방법을 알아보자

▼ 1) 메서드의 구조와 문법

- 메서드는 다음과 같은 구조로 선언됩니다:

```
[접근 제한자] [리턴 타입] [메서드 이름]([매개변수])
{
    // 메서드 실행 코드
}
```

- 접근 제한자(Access Modifier): 메서드에 접근할 수 있는 범위를 지정합니다. 주로 `public`, `private`, `protected` 등을 사용합니다.
- 리턴 타입(Return Type): 메서드가 반환하는 값의 데이터 타입을 지정합니다. 반환 값이 없을 경우 `void`를 사용합니다.
- 메서드 이름(Method Name): 메서드를 호출하기 위해 사용하는 이름입니다. 호출할 때 이 이름을 사용합니다.
- 매개변수(Parameters): 메서드에 전달되는 입력 값으로, 필요한 경우 0개 이상의 매개변수를 정의할 수 있습니다.
- 메서드 실행에 코드(Method Body): 중괄호(`{}`) 안에 메서드가 수행하는 작업을 구현하는 코드를 작성합니다.

예시

```
// 예시 1: 반환 값이 없는 메서드
public void SayHello()
{
    Console.WriteLine("안녕하세요!");
}

// 예시 2: 매개변수가 있는 메서드
public void GreetPerson(string name)
{
    Console.WriteLine("안녕하세요, " + name + "님!");
}

// 예시 3: 반환 값이 있는 메서드
public int AddNumbers(int a, int b)
{
    int sum = a + b;
```

```
    return sum;
}
```

▼ 2) 메서드 호출 방법

- 메서드를 호출하기 위해서는 메서드 이름과 필요한 매개변수를 전달하여 호출합니다.
- 메서드 호출은 다음과 같은 형식으로 이루어집니다:

```
[메서드 이름]([전달할 매개변수]);
```

- 예를 들어, `AddNumbers`라는 이름의 메서드를 호출하고자 할 때는 다음과 같이 호출합니다:

```
AddNumbers(10, 20);
```

- 호출 시 전달되는 매개변수는 메서드의 매개변수와 순서와 타입이 일치해야 합니다.
- 메서드는 호출되면 해당 메서드의 실행 코드를 수행하고, 필요한 경우 리턴 값이 있다면 반환합니다.

03. 매개변수와 반환값

✔️ 매개변수와 반환값을 활용하는 방법에 대해 알아보자

▼ 1) 매개변수의 개념과 활용

- 매개변수는 메서드에 전달되는 입력 값으로, 메서드 내에서 이 값을 활용하여 원하는 작업을 수행할 수 있습니다.
- 매개변수는 메서드의 선언부에 정의되며, 필요한 경우 0개 이상의 매개변수를 정의할 수 있습니다.
- 매개변수는 메서드 호출 시 전달되는 값에 따라 동적으로 결정되며, 호출 시에는 해당 매개변수의 값을 전달해야 합니다.

예제:

```
void PrintFullName(string firstName, string lastName)
{
    Console.WriteLine("Full Name: " + firstName + " " + lastName);
}

// 메서드 호출
PrintFullName("John", "Doe");
```

▼ 2) 반환값의 개념과 활용

- 반환값은 메서드가 수행한 작업의 결과를 호출자에게 반환하는 값입니다.
- 반환값은 메서드의 리턴 타입에 지정되며, 해당 타입의 값을 반환해야 합니다.
- 메서드 내에서 계산, 조작, 처리한 결과 등을 반환값으로 사용할 수 있습니다.

예제:

```
int AddNumbers(int a, int b)
{
    int sum = a + b;
    return sum;
}

// 메서드 호출 및 반환값 사용
int result = AddNumbers(10, 20);
Console.WriteLine("Sum: " + result);
```

▼ 3) void 형식과 반환값이 없는 메서드

- void는 메서드의 리턴 타입으로 사용되며, 해당 메서드가 값을 반환하지 않음을 나타냅니다.
- 반환값이 없는 메서드는 호출되면 메서드의 실행 코드를 수행하고 호출자에게 반환하지 않습니다.

예제:

```
void PrintMessage(string message)
{
    Console.WriteLine("Message: " + message);
}

// 메서드 호출
PrintMessage("Hello, World!");
```

▼ 4) 사용 예시

▼ **[코드스니펫] 메서드 기초**

```
void PrintLine()
{
    for (int i = 0; i < 10; i++)
    {
        Console.Write("=");
    }
    Console.WriteLine();
}

void PrintLine2(int count)
{
    for (int i = 0; i < count; i++)
    {
        Console.Write("=");
    }
    Console.WriteLine();
}

int Add(int a, int b)
{
    return a + b;
}


[사용 예시]
PrintLine();
PrintLine2(20);

int result = Add(10, 20);
Console.WriteLine(result);
```

04. 메서드 오버로딩

✔️ 같은 이름 다른 동작의 메서드 오버로딩!

▼ 1) 오버로딩 개념과 활용

- 메서드 오버로딩은 동일한 이름의 메서드를 다양한 매개변수 목록으로 다중 정의하는 개념입니다.

- 매개변수의 개수, 타입, 순서가 다른 여러 메서드를 동일한 이름으로 정의하여 메서드 호출 시 매개변수의 형태에 따라 적절한 메서드가 선택되도록 할 수 있습니다.
- 오버로딩은 메서드의 기능이나 작업은 동일하지만 입력값에 따라 다르게 동작해야 할 때 사용됩니다.

예제:

```
void PrintMessage(string message)
{
    Console.WriteLine("Message: " + message);
}

void PrintMessage(int number)
{
    Console.WriteLine("Number: " + number);
}

// 메서드 호출
PrintMessage("Hello, World!");  // 문자열 매개변수를 가진 메서드 호출
PrintMessage(10);  // 정수 매개변수를 가진 메서드 호출
```

▼ [코드스니펫] 오버로딩 기초

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

int AddNumbers(int a, int b, int c)
{
    return a + b + c;
}

// 메서드 호출
int sum1 = AddNumbers(10, 20);  // 두 개의 정수 매개변수를 가진 메서드 호출
int sum2 = AddNumbers(10, 20, 30);  // 세 개의 정수 매개변수를 가진 메서드 호출
```

05. 재귀 호출

✔️ 자칫 잘못하면 무한루프에 빠질수 있는 재귀 호출

▼ 1) 재귀 호출 개념과 동작 원리

- 재귀 호출은 메서드가 자기 자신을 호출하는 것을 의미합니다.
- 재귀 호출은 문제를 작은 부분으로 분할하여 해결하는 방법 중 하나로, 작은 부분의 해결 방법이 큰 문제의 해결 방법과 동일한 구조를 갖고 있는 경우에 적합합니다.
- 재귀 호출은 호출 스택에 호출된 메서드의 정보를 순차적으로 쌓고, 메서드가 반환되면서 스택에서 순차적으로 제거되는 방식으로 동작합니다.

▼ [코드스니펫] 재귀 호출 기초

```
void CountDown(int n)
{
    if (n <= 0)
    {
        Console.WriteLine("Done");
    }
    else
    {
        Console.WriteLine(n);
        CountDown(n - 1);  // 자기 자신을 호출
    }
}

// 메서드 호출
CountDown(5);
```

▼ 2) 재귀 호출의 활용과 주의점

- 재귀 호출은 복잡한 문제를 단순한 방식으로 해결할 수 있는 장점이 있습니다.
- 재귀 호출을 사용할 때 주의해야 할 점은 종료 조건을 명확히 정의해야 하며, 종료 조건을 만족하지 못하면 무한히 재귀 호출이 반복되어 스택 오버플로우 등의 오류가 발생할 수 있습니다.
- 재귀 호출은 메모리 사용량이 더 크고 실행 속도가 느릴 수 있으므로, 필요한 경우에만 적절히 사용하는 것이 좋습니다.

06. 메서드 활용 사례

▼ 1) 메서드를 사용한 코드의 재사용성

- 메서드는 일련의 작업을 수행하는 코드 블록으로, 반복적으로 사용되는 코드를 메서드로 분리함으로써 코드의 재사용성을 높일 수 있습니다.

- 동일한 작업을 수행하는 코드를 여러 곳에서 사용해야 할 때, 해당 작업을 메서드로 정의하고 필요한 곳에서 메서드를 호출하여 재사용할 수 있습니다.
- 이를 통해 중복 코드를 제거하고, 코드의 길이를 줄이고, 코드의 가독성과 유지보수성을 향상시킬 수 있습니다.

▼ 2) 메서드를 활용한 가독성과 유지보수성 개선

- 메서드는 코드의 일부분을 의미 있는 이름으로 추상화하고, 해당 메서드를 호출함으로써 코드의 의도를 명확히 전달할 수 있습니다.
- 긴 코드를 작은 단위로 분리하여 메서드로 정의하면, 코드가 간결해지고 가독성이 향상됩니다.
- 또한, 코드를 작은 단위로 분리하여 메서드로 관리하면, 유지보수가 용이해집니다. 특정 기능을 수정하거나 추가해야 할 때, 해당 메서드만 수정하면 되기 때문에 다른 부분에 영향을 주지 않고도 유지보수가 가능합니다.

▼ 3) 예제

```
// 원의 넓이를 계산하는 메서드
double CalculateCircleArea(double radius)
{
    double area = Math.PI * radius * radius;
    return area;
}

// 사각형의 넓이를 계산하는 메서드
double CalculateRectangleArea(double width, double height)
{
    double area = width * height;
    return area;
}

// 메서드 활용
double circleArea = CalculateCircleArea(5.0);
double rectangleArea = CalculateRectangleArea(3.0, 4.0);

Console.WriteLine("원의 넓이: " + circleArea);
Console.WriteLine("사각형의 넓이: " + rectangleArea);
```

07. 구조체

✔️ 사용자 정의 자료형의 시작

▼ 1) 구조체란?

- 여러 개의 데이터를 묶어서 하나의 사용자 정의 형식으로 만들기 위한 방법입니다.
- 구조체는 값 형식(Value Type)으로 분류되며, 데이터를 저장하고 필요한 기능을 제공할 수 있습니다.
- 구조체는 `struct` 키워드를 사용하여 선언합니다.
- 구조체의 멤버는 변수와 메서드로 구성될 수 있습니다.

```
struct Person
{
    public string Name;
    public int Age;

    public void PrintInfo()
    {
        Console.WriteLine($"Name: {Name}, Age: {Age}");
    }
}
```

▼ 2) 구조체의 사용

- 구조체는 변수를 선언하여 사용할 수 있습니다.
- 구조체의 멤버에는 접근할 때 `.` 연산자를 사용합니다.

```
Person person1;
person1.Name = "John";
person1.Age = 25;
person1.PrintInfo();
```

이전 강의

📕 2) 배열과 컬렉션

다음 강의

📕 1) 클래스와 객체