9강 상속과 다형성

매 강의 강의자료 시작에 PDF파일을 올려두었어요!

▼ PDF 파일

[수업 목표]

- C#에서 상속과 다형성의 개념을 이해하고 활용할 수 있습니다.
- 추상 클래스의 개념과 사용법을 이해하고 활용할 수 있습니다.

[목차]

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

---

01. 상속

상속(Inheritance)은 객체지향 프로그래밍에서 중요한 개념

▼ 1) 상속이란?

1. 상속의 개념:
    - 상속은 기존의 클래스(부모 클래스 또는 상위 클래스)를 확장하거나 재사용하여 새로운 클래스(자식 클래스 또는 하위 클래스)를 생성하는 것입니다.
    - 자식 클래스는 부모 클래스의 멤버(필드, 메서드, 프로퍼티 등)를 상속받아 사용할 수 있습니다.
    - 상속을 통해 부모 클래스의 기능을 확장하거나 수정하여 새로운 클래스를 정의할 수 있습니다.
2. 상속의 장점:
    - 코드의 재사용성: 상속을 통해 기존 클래스의 코드를 재사용할 수 있으므로, 반복적인 코드 작성을 줄일 수 있습니다.
    - 계층 구조의 표현: 클래스 간의 계층 구조를 표현하여 코드의 구조를 명확하게 표현할 수 있습니다.
    - 유지보수성의 향상: 상속을 통해 기존 클래스의 수정이 필요한 경우, 해당 클래스만 수정하면 됩니다. 이로써 코드의 유지보수성이 향상됩니다.
3. 상속의 종류:
    - 단일 상속: 하나의 자식 클래스가 하나의 부모 클래스만 상속받는 것을 말합니다. C#에서는 단일 상속만을 지원합니다.
    - 다중 상속: 하나의 자식 클래스가 여러 개의 부모 클래스를 동시에 상속받는 것을 말합니다. C#은 다중 상속을 지원하지 않습니다.
    - 인터페이스 상속: 클래스가 인터페이스를 상속받는 것을 말합니다. 인터페이스는 다중 상속을 지원하며, 클래스는 하나의 클래스와 여러 개의 인터페이스를 동시에 상속받을 수 있습니다.
4. 상속의 특징:
    - 부모 클래스의 멤버에 접근:
        - 자식 클래스는 상속받은 부모 클래스의 멤버에 접근할 수 있으며, 이를 통해 부모 클래스의 기능을 재사용할 수 있습니다.
    - 메서드 재정의:
        - 자식 클래스는 부모 클래스의 메서드를 재정의하여 자신에게 맞게 수정할 수 있습니다. 이를 통해 다형성(Polymorphism)을 구현할 수 있습니다.
    - 상속의 깊이 :

- 클래스는 다수의 계층적인 상속 구조를 가질 수 있습니다. 부모 클래스가 또 다른 클래스의 자식 클래스가 될 수 있으며, 이를 통해 상속의 계층 구조를 형성할 수 있습니다.
- 상속의 깊이가 깊어질수록 클래스 간의 관계가 복잡해질 수 있으므로, 적절한 상속의 깊이를 유지하고 상속을 적절하게 사용하는 것이 중요합니다.

5. 접근 제한자와 상속:

- 상속 관계에서 멤버의 접근 제한자는 중요한 역할을 합니다. 부모 클래스의 멤버의 접근 제한자에 따라 자식 클래스에서 해당 멤버에 접근할 수 있는 범위가 결정됩니다.
- 접근 제한자를 통해 상속된 멤버의 가시성을 조절하여 캡슐화와 정보 은닉을 구현할 수 있습니다.

▼ 2) 예제 코드

▼ [코드스니펫] 상속 기초

```
// 부모 클래스
public class Animal
{
    public string Name { get; set; }
    public int Age { get; set; }

    public void Eat()
    {
        Console.WriteLine("Animal is eating.");
    }

    public void Sleep()
    {
        Console.WriteLine("Animal is sleeping.");
    }
}

// 자식 클래스
public class Dog
{

}

public class Cat
{

}
```

```
// 부모 클래스
public class Animal
```

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

    public void Eat()
    {
        Console.WriteLine("Animal is eating.");
    }

    public void Sleep()
    {
        Console.WriteLine("Animal is sleeping.");
    }
}

// 자식 클래스
public class Dog : Animal
{
    public void Bark()
    {
        Console.WriteLine("Dog is bark.");
    }
}

public class Cat : Animal
{
    public void Sleep()
    {
        Console.WriteLine("Cat is sleeping.");
    }

    public void Meow()
    {
        Console.WriteLine("Cat is meow.");
    }
}

// 사용 예시
Dog dog = new Dog();
dog.Name = "Bobby";
dog.Age = 3;

dog.Eat();      // Animal is eating.
dog.Sleep();    // Animal is sleeping.
dog.Bark();     // Dog is barking

Cat cat = new Cat();
cat.Name = "KKami";
cat.Age = 10;

cat.Eat();
cat.Sleep();
cat.Meow();
```

02. 다형성

✔️ 같은 타입이지만 다양한 동작을 수행할 수 있는 능력

▼ 1) 가상(Virtual) 메서드

- 가상 메서드는 기본적으로 부모 클래스에서 정의되고 자식 클래스에서 재정의할 수 있는 메서드입니다.
- 가상 메서드는 `virtual` 키워드를 사용하여 선언되며, 자식 클래스에서 필요에 따라 재정의될 수 있습니다.
- 이를 통해 자식 클래스에서 부모 클래스의 메서드를 변경하거나 확장할 수 있습니다.

▼ [코드스니펫] 가상 메서드 기초

```
public class Unit
{
    public void Move()
    {
        Console.WriteLine("두발로 걷기");
    }

    public void Attack()
    {
        Console.WriteLine("Unit 공격");
    }
}

public class Marine
{

}

public class Zergling
{

}
```

```
public class Unit
{
    public virtual void Move()
    {
        Console.WriteLine("두발로 걷기");
    }

    public void Attack()
```

```
    {
        Console.WriteLine("Unit 공격");
    }
}

public class Marine : Unit
{

}

public class Zergling : Unit
{
    public override void Move()
    {
        Console.WriteLine("네발로 걷기");
    }
}
```

```
// 사용 예시
// #1 참조형태와 실형태가 같을때
Marine marine = new Marine();
marine.Move();
marine.Attack();

Zergling zergling = new Zergling();
zergling.Move();
zergling.Attack();

// #2 참조형태와 실형태가 다를때
List<Unit> list = new List<Unit>();
list.Add(new Marine());
list.Add(new Zergling());

foreach (Unit unit in list)
{
    unit.Move();
}
```

▼ 2) 추상(Abstract) 클래스와 메서드

- 추상 클래스는 직접적으로 인스턴스를 생성할 수 없는 클래스
- 주로 상속을 위한 베이스 클래스로 사용됩니다.
- 추상 클래스는 `abstract` 키워드를 사용하여 선언되며, 추상 메서드를 포함할 수 있습니다.

- 추상 메서드는 구현부가 없는 메서드로, 자식 클래스에서 반드시 구현되어야 합니다.

▼ [코드스니펫] 추상클래스 기초

```
abstract class Shape
{
    public abstract void Draw();
}

class Circle
{

}

class Square
{

}

class Triangle
{

}
```

```
abstract class Shape
{
    public abstract void Draw();
}

class Circle : Shape
{
    public override void Draw()
    {
        Console.WriteLine("Drawing a circle");
    }
}

class Square : Shape
{
    public override void Draw()
    {
        Console.WriteLine("Drawing a square");
    }
}

class Triangle : Shape
{
    public override void Draw()
    {
        Console.WriteLine("Drawing a triangle");
```

```
    }
}
```

```
List<Shape> list = new List<Shape>();
list.Add(new Circle());
list.Add(new Square());
list.Add(new Triangle());

foreach (Shape shape in list )
{
    shape.Draw();
}
```

▼ 3) 오버라이딩 과 오버로딩

오버라이딩(Overriding)과 오버로딩(Overloading)은 객체 지향 프로그래밍에서 다른 개념을 나타내는 두 용어입니다.

- 오버라이딩(Overriding)
  - 부모 클래스에서 이미 정의된 메서드를 자식 클래스에서 재정의하는 것을 의미합니다.
  - 이는 상속 관계에 있는 클래스 간에 발생하며, 메서드의 이름, 매개변수 및 반환 타입이 동일해야 합니다.
  - 오버라이딩을 통해 자식 클래스는 부모 클래스의 메서드를 재정의하여 자신에게 맞는 동작을 구현할 수 있습니다.

```
public class Shape
{
    public virtual void Draw()
    {
        Console.WriteLine("Drawing a shape.");
    }
}

public class Circle : Shape
{
    public override void Draw()
    {
        Console.WriteLine("Drawing a circle.");
    }
}

public class Rectangle : Shape
```

```
{
    public override void Draw()
    {
        Console.WriteLine("Drawing a rectangle.");
    }
}

Shape shape1 = new Circle();
Shape shape2 = new Rectangle();

shape1.Draw();  // Drawing a circle.
shape2.Draw();  // Drawing a rectangle.
```

- 오버로딩(Overloading)
  - 동일한 메서드 이름을 가지고 있지만, 매개변수의 개수, 타입 또는 순서가 다른 여러 개의 메서드를 정의하는 것을 의미합니다.
  - 오버로딩을 통해 동일한 이름을 가진 메서드를 다양한 매개변수 조합으로 호출할 수 있습니다.

```
public class Calculator
{
    public int Add(int a, int b)
    {
        return a + b;
    }

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

Calculator calc = new Calculator();
int result1 = calc.Add(2, 3);         // 5
int result2 = calc.Add(2, 3, 4);      // 9
```

이전 강의

📕 1) 클래스와 객체

다음 강의

10강 고급 문법 및 기능