C# w2

king 14기 이서우

▼ 객체 지향 프로그래밍

모든 데이터를 오브젝트(object;물체)로 취급하여 프로그래밍 하는 방법으로, 처리 요구를 받은 객체가 자기 자신의 안에 있는 내용을 가지고 처리하는 방식

▼ 객체 지향 프로그래밍의 특징

1. **추상화**
   객체의 공통적인 속성, 기능을 추출해 정의하는 것으로 인터페이스와 추상 클래스가 있다. 각기 다른 클래스의 객체들이 공통적인 기능을 가진다면 그 기능을 묶여서 관리하는 것이 편리하다. 객체 지향 프로그래밍에서는 역할/구현을 구분하는데, 그 중에서 '역할'만 담당한다.
2. **캡슐화**
   클래스, 함수 등을 통해 기능을 하나의 단위로 묶는 것. 데이터, 함수에 접근을 제한하여 은닉성을 높일 수 있다. 외부 사용자는 클래스의 내부 구조를 몰라도 다룰 수 있어 대규모 개발에서 중요한 특징이다.
3. **상속성**
   부모 클래스를 상속 받아 자식 클래스에서 재사용할 수 있다. 비슷한 내용의 코드가 반복되면 코드의 길이가 길어지는 것을 방지할 수 있다.
   a. **base**
      base는 기반 클래스를 가리키는 키워드이다. base 키워드를 통해 기반클래스의 멤버에 접근할 수 있다.
   b. **sealed 한정자**
      클래스를 sealed 한정자를 이용해 선언하면 상속할 수 없는 클래스가 된다.
   c. **base()**
      기반 클래스/부모클래스의 생성자
      public Derived(string Name) : base(Name)
4. **다형성**
   하나의 객체가 여러 형태를 지니는 것이다. 오버로딩, 오버라이딩, 인터페이스 등 똑

같은 이름의 메소드여도 다른 방식으로 작동함으로써 코드의 수정을 최소화하고 길이를 줄일 수 있다.

a. **오버라이딩 (overriding)**
상위 클래스가 가지고 있는 메서드를 하위 클래스가 재정의해서 사용.
클래스 타입에 따라 다른 메소드가 실행된다.

`virtual`: 부모 클래스의 메소드에 virtual을 붙여준다.

`override`: 자식 클래스의 메소드에 override를 붙여준다.

▼ 클래스와 객체, 인스턴스

`클래스` : 객체에 속성과 기능을 넣어줄 설계도

`객체` : 소프트웨어 세계에 구현할 대상이며 속성과 기능을 가지는 프로그램 단위

`인스턴스` : 클래스에 따라 메모리상에 구현된 실체

▼ 접근한정자

1. `public`
모든 외부(파생클래스 포함)에서 이 타입(Type: 클래스, 구조체, 인터페이스, 델리게이트 등)을 엑세스할 수 있다.
(개별 타입 멤버의 엑세스 권한은 해당 멤버의 접근 제한자에 따라 별도로 제한될 수 있다)
2. `internal`
동일한 Assembly 내에 있는 다른 타입들이 엑세스 할 수 있다. 하지만, 다른 어셈블리에서는 접근이 불가하다.
3. `protected`
파생클래스에서 이 클래스 멤버를 엑세스할 수 있다.
4. `private`
동일 클래스/구조체 내의 멤버만 접근 가능하다.

▼ is/as 키워드

1. `is`
객체가 특정 타입인지 검사한다.

```
if (obj is MyClass) { ... }
```

2. as
   객체를 특정 타입으로 변환하며, 변환에 실패하면 'null'을 반환한다.

```
MyClass myObj = obj as MyClass;
```

▼ **추상 클래스 (abstract class)**

객체 지향 프로그래밍에서 공통적인 특성과 메서드를 정의하고, 구체적인 구현은 서브 클래스에서 이루어지도록 설계된 클래스이다. 추상 클래스는 인스턴스를 만들 수 없고, 다른 클래스가 상속하여 구현을 완성해야 한다.

▼ **인터페이스 (interface)**

클래스나 구조체 등의 형식(type)에 대한 계약(contract)을 정의하는 데 사용된다. 인터페이스의 모든 멤버는 구체적인 구현(implement)을 제공하지 않는 추상 메서드, 속성, 이벤트를 선언하며, 해당 클래스나 구조체에서 이 인터페이스의 멤버를 구체적으로 정의하여 구현해야 한다.

▼ **델리게이트 (delegate)**

자기에게 전달된 함수를 대신 실행시켜주는 역할을 하는 함수 포인터
→ 자신에게 할당된 함수를 대신 실행
하나의 형식(type)으로, 메서드에 대한 참조이다. 값이 아닌 '코드'자체를 넘기고 싶을 때 사용할 수 있다.

- **액션 (action)**
  반환 타입이 void인 메소드를 위해 특별히 설게된 제네릭 델리게이트
  액션은 사용자 정의 델리게이트 대신 사용할 수 있어 코드를 간소화하고 표현력을 높일 수 있다. 매개변수가 없는 액션은 <>을 사용하지 않는다.

- **이벤트 (event)**
  프로그래밍에 어떤 일이 생겼을 때 실행되는 객체
  c#에서 발생하는 특정한 동작을 나타내는 신호로 해당 동작에 대한 응답을 받을 수 있도

록 프로그램에 알림을 제공한다. 이벤트는 객체 간의 상호작용에서 중요한 역할을 수행하며, 일종의 통신 메커니즘으로 볼 수 있다. 이벤트는 발생한 사건을 처리하기 위해 이벤트 핸들러라고 하는 메서드를 등록하고, 이벤트가 발생하면 등록된 모든 이벤트 핸들러가 순차적으로 호출되어 해당 동작에 대한 응답을 처리한다. 이벤트 핸들러는 이벤트에 대한 구독자로서, 이벤트가 발생하면 자동으로 호출되어 처리 작업을 수행한다.

▼ 제네릭 (generic)

코드의 재사용성과 유연성을 향상해 주는 도구
데이터 형식을 일반화하여 재사용 가능한 코드를 작성할 수 있도록 도와준다.
제네릭을 사용하면 다양한 형식의 데이터를 처리하는 메서드와 클래스를 작성할 수 있으며 컴파일 시점에서 안정성을 보장해 준다.