C#

킹 14 기 이민경

2466040

1. class

인스턴스 생성 시 개체의 모습에 대한 템플릿. 클래스는 위젯 모습을 나타낸 틀과 같다. 개체는 클래스의 인스턴스이고, 클래스라는 틀을 이용해 만든 위젯이라고 할 수 있다. 개체가 클래스의 인스턴스이기 때문에 클래스에서 개체를 만드는 과정을 인스턴스화라고 한다.

2. 객체지향 프로그래밍

객체지향 프로그래밍은 프로그램을 수많은 '객체'라는 기본 단위로 나누고 이들의 상호작용으로 서술하는 프로그래밍 방식이다.

1) **캡슐화**

   캡슐화로 상세 내용을 숨길 수 있다. 필요할 때는 여전히 세부 구현에 접근할 수 있지만, 세부 구현을 캡슐화 하면 큰 프로그램을 이해하기 더 쉽게 만들며, 부주의한 수정을 막을 수 있고, 코드 변경으로 인한 영향을 캡슐화 범위 내로 제한하기 때문에 코드의 유지 관리가 더 쉬워진다. 캡슐화의 예시로 메서드가 있다.

   캡슐화 하는 작업은 메서드와 데이터를 함께 한 개체로 캡슐화 하는 것이다. 이는 더이상 각각 다룰 필요가 없도록 클래스 멤버를 모두 모은 것이다.

2) **상속**

   객체지향 프로그래밍에서 상속을 다르게 표현하면 "~의 일종" 이라고 표현할 수 있다. 상속은 클래스 계층 구조를 정의하는 것이다. 예를 들어 음식이라는 클래스를 만들었다고 하자. 음식이라는 클래스 아래 한식, 일식, 중식, 양식 등으로 구분되는 클래스를 만들 수 있고, 이 하위 클래스들은 '음식' 이라는 클래스의 속성을 전부 갖는다. '한식'이라는

클래스 아래 더 구체화된 '비빔밥', '양념갈비', '냉면' 등의 클래스를 만들 수 있고 이들은 '한식'이 갖는 모든 속성을 물려받는다. 이는 한식이 물려받은 '음식'이라는 클래스의 속성도 포함되어있다. 더 구체화된 형식을 **파생형식** 또는 **하위형식**이라 하고 더 일반적인 형식을 **기본형식** 또는 **상위형식**이라고 한다. 여기서 기본형식은 '음식'이 되겠고 파생형식은 한식 일식 중식에 해당된다. 파생형식을 사용하면 해당 클래스가 기본 형식보다 더 구체성을 갖는 형식을 만들수 있다.

3) 다형성

다형성은 하나의 메서드나 형식에서 다양한 형태를 구현하고 있음을 뜻한다. 이는 상속받은 자식 클래스가 다양한 타입을 가질 수 있는 것을 뜻한다.

(1) 오버로딩

메서드의 이름은 같고 매개변수의 유형과 개수가 다르도록 하는 것을 의미한다. 리턴값만 다르게 만들 수 없다. 즉, 오버로딩은 기존에 없던 새로운 메서드를 정의하는 것이다. **오버로딩은 같은 기능을 하는 메서드를 하나의 이름으로 표현할 수 있다는 장점이 있다.** 그 예시로 println() 메서드가 있다. println() 메서드가 오버로딩이 가능하기에 int 형 인자, string 형 인자, char 형 인자를 모두 받아서 동작할 수 있다. **오버로딩은 모든 접근 제어자를 사용할 수 있다.** **같은 클래스** 내에서 적용된다.

(2) 오버라이딩

상위 클래스가 가지고 있는 메서드를 하위 클래스가 재정의해서 사용하는 것을 의미한다. 메서드의 이름은 물론 파라미터의 개수나 타입도 동일해야 하며, 주로 상위 클래스의 동작을 상속받는 하위 클래스에서 변경하기 위해 사용된다. 즉, 오버라이딩은 상속받은 메서드의 내용만 변경하는 것이다. **부모 클래스를 상속받은 자식 클래스는 부모의 메서드도 전부 상속 받는데 자식 클래스에서 부모에게 상속받은 메서드를 변경할 수 있다. 그러나 자식 클래스에서 오버라이딩하는 메소드의 접근 제어자(public, private, protected)는 부모 클래스보다 좁게 설정할 수 없다.** 예외는 부모 클래스의 메소드보다 많이 선언할 수 없다. **상속관계**에서 적용된다.

3. as/is keyword

1) is

is 키워드는 특정 객체 타입과 호환이 가능한지 확인하는 연산자이다. 호환 가능성을 bool 값으로 나타내준다.

2) as

as 키워드는 형식 변환 연산 시 사용된다. 이때 변환이 가능하지 않은 경우 null 이 반환된다.

4. abstract class

추상 클래스는 추상 엔티티를 나타낸다. 추상 멤버는 추상 엔티티에서 파생된 개체가 포함해야 하는 것을 정의하지만 구현을 포함하지는 않는다. 추상 클래스 내에서 많은 기능이 구현되지 않는 경우도 종종 있다. 하지만 추상 클래스에서 하위 형식을 파생시킬 때는 추상 기본 클래스의 해당 추상 메서드에 대한 구현을 제공해야 한다.

5. interface

인터페이스는 추상 클래스와 달리 제공하는 서비스와 상세 구현을 완전히 분리할 수 있다. 인터페이스는 구현을 공유할 필요 없이 멤버 시그니처를 공유할 수 있다는 점이다. 예를 들어, 전원을 공급하면 열을 발생시키는 열원장치가 있다고 하자. 이때 전원만 공급되면 열을 발생시켜주는 것을 인터페이스라고 할 수 있다. 그 전원이 태양열이던, 화학 에너지이던, 핵 에너지이던 상관없이 '전원'이라는 조건이 충족되면 '열을 발생시킨다' 라는 작업을 해준다는 것 이다.

6. generic

일반적으로 클래스를 정의할 때, 클래스 내의 모든 데이터 타입을 지정해줘야 한다. 그러나 데이터 타입만 다르고 다른 부분이 동일한 클래스의 경우, 제네릭을 통해 추가

클래스를 만들 필요 없이 간소화 해줄 수 있다. 이때 형식 매개변수(type parameter)을 지정해 만들 수 있다.

7. delegate

c,c++에서의 '함수 포인터'와 비슷한 개념이다. 대리자를 이용하면 개체를 참조하듯이 메서드를 참조할 수 있고, 이렇게 획득한 메서드를 일반적인 메서드처럼 호출할 수도 있다. 델리게이트를 사용하면 함수를 변수처럼 사용할 수 있게 된다.

1) 액션

Action 은 델리게이트의 일종이다. 다만 delegate 와 달리 action 은 반환 형식이 없는 함수만 참조할 수 있다. 액션 대리자는 사용자 지정 대리자를 명시적으로 선언하지 않고 메서드를 매개 변수로 전달 하도록 대리자를 사용할 수 있다. 캡슐화 된 메서드에 이 대리자에 의해 정의되는 메서드 시그니처와 일치 해야 한다.

8. event

이벤트는 개체에서 작업 실행을 알리기 위해 보내는 메세지이다. 이 작업은 버튼 클릭과 같은 사용자 조작으로 발생하거나, 속성 값 변경 등의 다른 프로그램 논리에서 발생할 수 있다. 이벤트를 발생시키는 개체를 **이벤트 전송자**라고 하며 이벤트 전송자는 어떤 개체 또는 메서드가 발생되는 이벤트를 어떤 객체나 메서드가 받을지 모른다.