is / as

```
class ClassA{}
class ClassB{}
static void main(string[] args)
{
        object a = new ClassA();
        ClassB b = a as ClassB;
```

as 키워드는 형식 변환 연산에서 사용되는데, 캐스트 연산과 달리 변환이 가능하지 않은 경우 null을 반환시킨다. 참조 변환에서 사용된다.

```
class ClassA{}
class ClassB{}
static void main(string[] args)
{
        object a = new ClassA();
        ClassB b;
        if(a is ClassB) b= a as ClassB;
}
```

is 연산자는 특정 객체와의 호환이 가능한지 확인하는 연산자이다. 가능할 경우 true를 반환하고(위 코드에서는 b = a as ClassB를 실행) 아닐 경우 false를 반환한다.

abstract 클래스

추상 클래스(abstract class)는 미완성된 클래스로, 추상 메소드(미완성 메소드)를 포함하고 있다는 것을 뜻한다. abstract 클래스는 자식 클래스에서 공유할 수 있도록 부모 클래스(abstract 클래스)의 공통 정의를 제공하는 것이다. 추상 클래스는 new 연산자를 이용하여 인스턴스를 생성할 수 없고, 오직 자식 클래스에서 상속을 통해서만 구현이 가능하다. 또, 추상 메소드는 abstract 클래스에서만 선언할 수 있고, static 또는 virtual 키워드를 사용할 수 없다. 추상 메소드는 가상 메소드에 속한다.

```
abstract class ClassA{
        public string name;
        public abstract void Move();
        public void Move2(){
        Console.WriteLine("..");}
}
class ClassB : ClassA
{       public override void Move(){
        Console.WriteLine("...");}
}
```

인터페이스

인터페이스 역시 abstract 클래스와 같이 불완전 클래스이며 abstract 클래스보다 추상화 정도가 높다. 인터페이스는 다른 클래스를 작성하는 것을 돕기 위해 작성되며, 내부의 모든 메소드가 가상 메소드로 간주된다. 따라서 virtual 키워드를 지정하지 못 한다. 모두 public을 통해 선언되며, 개발 시간을 단축시키기 위해 사용된다.

```
interface   ClassA{
          void Writelog(string log);}
class ClassB  : ClassA{
          public void Writelog(string log){
                    Console.WriteLine("..");}
}
```

abstract 클래스와는 달리 다중 상속을 할 수 있다.

델리케이트

c/c++의 포인터와 비슷한 개념으로, 메서드를 매개 변수로 전달할 수 있다.

```
delegate int Test(object A);
```

위와 같은 방식으로 사용하는 형식이다.
델리케이트를 이용하면 메서드의 파라미터로 메소드를 전달할 수 있고, 델리케이트 자체는 형식이며 객체를 생성하여 사용한다. 델리게이트 객체는 여러 함수를 등록할 수 있고 제거할 수 있다. 델리게이트 체인을 이용하여 익명 메소드 호출도 가능하며 이는 일회성으로 사용될 코드를 간단히 작성하기 위해 사용된다. 또한, 콜백을 위해 자주 사용된다.

액션

액션은 C#의 내장된 델리게이트 중 하나로, 반환값이 없는 메소드를 참조할 수 잇ㄴ느 타입으로 매개변수를 받아서 반환값이 없는 메소드를 참조하는 기능을 한다. 즉, 무명 메소드를 만들 때 자주 사용된다.
이와 비슷하게 c#에 내장된 델리게이트 중 하나로 Func이 있다. 이 둘은 c#을 이용할 때 별도의 선언이 필요 없고, func은 반환 값이 있다는 차이가 있다.

이벤트

어떠한 일이 생겼을 때 알려주는 객체가 필요할 때가 있는데, 이러한 객체를 만들 때 이벤트를 사용한다. 이는 델리게이트와 비슷한 원리로 작동한다. (델리게이트의 확장개념과 비슷하게 볼 수 있다.)

```
public event ABC(델리게이트 명)
```

다만 이벤트는 델리게이트와는 달리 인터페이스 내부에서 선언할 수 있고, public으로 선언되어 있어도 클래스 외부에서 호출할 수 없다. 따라서 이벤트는 객체의 상태 변화 및 사건의 발생을 알리기 위한 용

도로 사용된다.

제네릭

제네릭을 사용하면 코드의 재사용성과 유연성을 향상시켜준다. 제네릭은 데이터 형식을 일반화하여 재사용 가능한 코드를 작성하게끔 돕는다. 이는 주로 여러 데이터 형식에서 동일한 로직을 적용해야 할 때, 컬렉션 타입에서 다양한 데이터 형식을 저장하고 관리해야 할 때, 데이터 형식에 따라 다른 연산을 수행해야 할 때 사용된다.