

추상 클래스

UPDATE 2023

01. 추상 클래스 개요

- 추상 클래스(Abstract Class)는 인터페이스와 클래스의 중간 단계로 인터페이스와 달리 구현은 할 수 있지만 인스턴스 생성이 불가능하다.
- 클래스 성격에 가까워 다중 상속은 불가능하며 단일 상속만 가능하다.
- 추상 클래스의 각 멤버는 public 외에 접근 한정자를 갖을 수 있으며 생략시 클래스와 같이 private로 선언된다. 또한 필드, 생성자, 메서드등을 갖을 수 있다.
- 추상 클래스는 일반적인 클래스들의 부모 역할을 하는 클래스, 즉 공통 기능들을 모아 놓은 클래스 역할을 하며 다른 클래스에 상속을 준 후 추가 기능을 하위 클래스에 구현하도록 하는 강제성을 띈다.
- 클래스를 설계할 때 부모 클래스 역할을 하면서 강제로 자식 클래스에 특정 멤버 이름을 물려줄 때 사용한다.

02. 추상 클래스 선언 및 사용

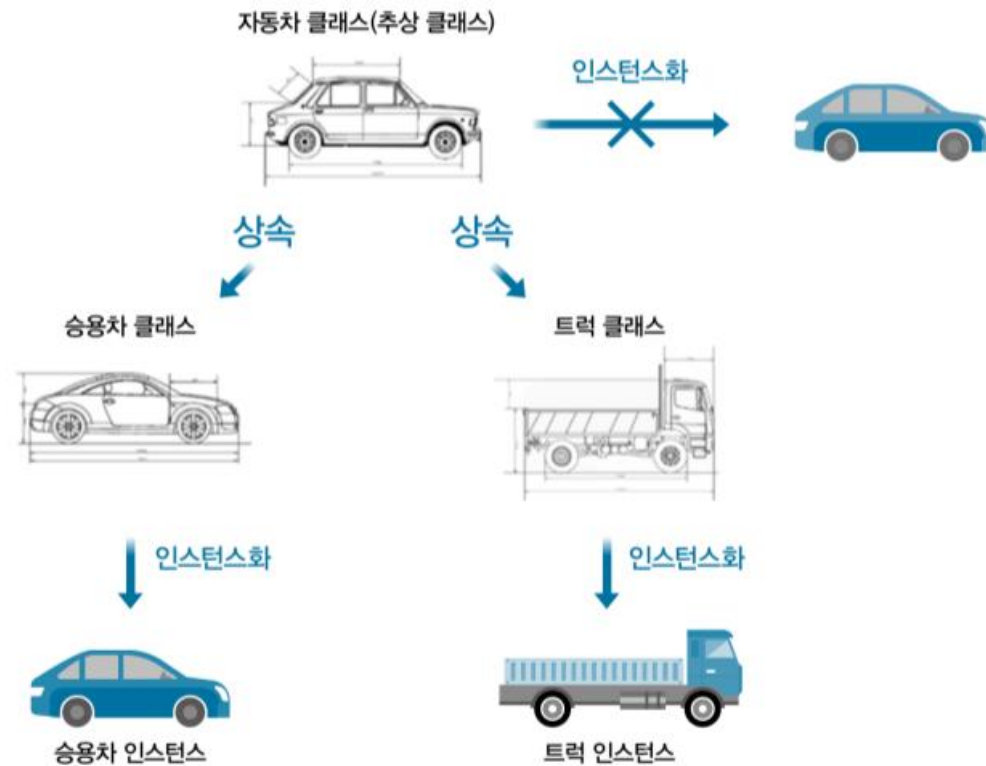
- 추상클래스 선언

```
abstract class 클래스명  
{  
    // 명령어  
}
```

추상 클래스로 지정할 때는
클래스 이름 앞에 abstract
키워드를 붙이면 된다

```
public abstract class Shape  
{  
    참조 4개  
    public abstract double GetArea();  
}
```

02. 추상 클래스 선언 및 사용



02. 추상 클래스 선언 및 사용

참조 2개

```
public abstract class Car  
{
```

```
    int speed = 0;
```

참조 0개

```
    void UpSpeed(int speed)
```

```
{
```

```
        this.speed += speed;
```

```
}
```

```
}
```

02. 추상 클래스 선언 및 사용

참조 2개

```
public abstract class Car
{
    int speed = 0;

    참조 0개
    void UpSpeed(int speed)
    {
        this.speed += speed;
    }
}
```

참조 0개

```
public class Sedan : Car
{
}
```

참조 0개

```
public class Truck : Car
{
}
```

02. 추상 클래스 선언 및 사용

```
// Car car1 = new Car(); // 오류 발생
Sedan sedan1 = new Sedan();
Console.WriteLine("\n\t 승용차 인스턴스 생성~~~");
Truck truck1 = new Truck();
Console.WriteLine("\n\t 트럭 인스턴스 생성~~~");
```

03. 추상 메소드

- 추상 메소드는 추상 클래스가 인터페이스 역할을 하기 위한 장치로 상속 받는 자식(파생) 클래스에서 메소드의 구현을 필수로 할 수 있도록 한다.
- 추상 메서드(Abstract Method)는 메서드 앞에 `abstract`를 붙인 것이다.
- 추상 메서드는 본체 코드가 존재하지 않는다.

```
abstract 반환형 메서드_이름(파라미터) ;
```

```
abstract double GetArea();
```


03. 추상 메소드

- 기본 클래스를 상속받은 파생 클래스에서는 override를 붙여 추상 메서드를 구현할 수 있다.

```
override void Work( )  
{  
    // 필요한 내용을 구현  
}
```

```
abstract class AbstractBase  
{  
    public abstract void SomeMethod();  
}  
  
class Derived : AbstractBase  
{  
    public override void SomeMethod()  
    {  
        // Something  
    }  
}
```

03. 추상 메소드

- 기본 클래스를 상속받은 파생 클래스에서는 override를 붙여 추상 메서드를 구현할 수 있다.

```
override void Work( )  
{  
    // 필요한 내용을 구현  
}
```

```
abstract class AbstractBase  
{  
    public abstract void SomeMethod();  
}  
  
class Derived : AbstractBase  
{  
    public override void SomeMethod()  
    {  
        // Something  
    }  
}
```

03. 추상 메소드

```
public abstract class Shape
```

```
{  
  .....  
}
```

2

참조 4개

```
public abstract double GetArea();
```

참조 4개

```
public class Square : Shape
```

```
{
```

```
    private int Size;
```

참조 2개

```
    public Square(int size)
```

```
    {
```

```
        this.Size = size;
```

```
    }
```

참조 4개

```
    public override double GetArea()
```

```
    {
```

```
        return this.Size * this.Size;
```

```
    }
```

```
}
```

03. 추상 메소드

```
public class Circle : Shape
{
    private double Radius;

    참조 1개
    public Circle(double radius)
    {
        this.Radius = radius;
    }

    참조 2개
    public override double GetArea()
    {
        return this.Radius * this.Radius * 3.14;
    }
}
```

03. 추상 메소드

```
Square square = new Square(10);  
Console.WriteLine(square.GetArea());
```

```
Circle circle = new Circle(5.5);  
Console.WriteLine(circle.GetArea());
```

```
100  
94.985
```

03. 추상 메소드

```
abstract class Car2
{
    public int speed = 0;

    참조 2개
    public void UpSpeed(int speed)
    {
        this.speed += speed;
    }

    참조 4개
    public abstract void Work();
}
```

03. 추상 메소드

```
class Sedan2 : Car2
{
    참조 2개
    public override void Work()
    {
        Console.WriteLine("승용차가 사람을 태우고 있습니다.");
    }
}
```

```
참조 2개
class Truck2 : Car2
{
    참조 2개
    public override void Work()
    {
        Console.WriteLine("트럭이 짐을 싣고 있습니다.");
    }
}
```

03. 추상 메소드

```
Sedan2 sedan2 = new Sedan2();  
sedan2.Work();  
sedan2.UpSpeed(100);  
Console.WriteLine("\t sedan2.speed : " + sedan2.speed);  
Console.WriteLine();
```

```
Truck2 truck2 = new Truck2();  
truck2.Work();  
.....  
truck2.UpSpeed(200);  
.....  
Console.WriteLine("\t truck2.speed : " + truck2.speed);  
.....
```

승용차가 사람을 태우고 있습니다.

sedan2.speed : 100

트럭이 짐을 싣고 있습니다.

truck2.speed : 200

퀴즈

- Animal 추상 클래스를 이용하여 자식 클래스 Tiger, Cat 클래스를 만들고 객체화 하여 출력결과를 확인하여라

```
abstract class Animal
{
    참조 2개
    public void Run()
    {
        Console.WriteLine($"{\t 엄청 빨리 달린다");
    }

    참조 4개
    public abstract void Hunt(string What, string Where);
    참조 4개
    public abstract void Sleep(int time, string Where);
}
```

퀴즈

참조 2개

```
class Tiger : Animal
```

```
{
```

```
}
```

코드 작성

퀴즈

```
class Cat : Animal
```

```
{
```

코드 작성

```
}
```

퀴즈

```
Tiger tiger = new Tiger();
```

코드 작성

```
Console.WriteLine();  
Cat cat = new Cat();
```

코드 작성

엄청 빨리 달린다
호랑이가 7시간을 동굴에서 잤다.
호랑이가 바다에서 물고기를/를 사냥한다.

엄청 빨리 달린다
고양이가 3시간을 침대에서 잤다.
고양이가 공원에서 쥐를/를 사냥한다.

인터페이스

Update 2023

01. 인터페이스 개요

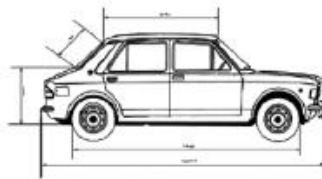
- 인터페이스는 클래스 또는 구조체에 포함될 수 있는 관련 메서드들을 묶어 관리하는 용도로 사용한다.
- 인터페이스는 명세서(specification) 으로 클래스가 따라야 하는 약속에 해당한다.
- 인터페이스는 interface 키워드를 사용하여 정의하며 실행 가능한 코드와 데이터를 포함하고 있지는 않다.
- 인터페이스에는 메서드, 속성 등을 정의할 수 있으며 단일 상속만 지원하는 클래스와 달리 인터페이스는 다중 상속이 가능하다.

01. 인터페이스 개요

- 인터페이스 멤버는 액세스 한정자를 붙이지 않으며 항상 public이고, virtual 및 static을 붙일 수 없다.
- C# 에서 인터페이스 이름은 ICar, IFood, IComputer 형태로 대문자 I로 시작한다.
- 인터페이스는 인스턴스화 되지 않으며 인터페이스를 상속받는 파생 클래스(자식 클래스)를 이용하여 인스턴스화 된다.
- 인터페이스는 계약(contract) 의미가 강하며 메서드등을 이용하여 구조를 미리 정의할 때 사용한다.

02. 인터페이스 선언 및 사용

자동차 클래스<추상 클래스>



필드

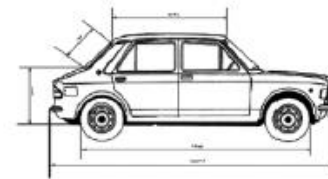


추상 메서드



일반 메서드

자동차 인터페이스



필드



추상 메서드



일반 메서드

- 인터페이스 선언

interface I인터페이스명

{

// 추상메서드

반환형식 메소드이름(매개변수목록);

}

```
interface ILogger
```

```
{
```

```
    void WriteLog( string log );
```

```
}
```


02. 인터페이스 선언 및 사용

- 인터페이스를 상속받는 파생 클래스 선언 : 인터페이스에 선언된 메소드는 반드시 구현해야 한다.

```
class ConsoleLogger : ILogger
{
    public void WriteLog(string message )
    {
        Console.WriteLine(
            "{0} {1}",
            DateTime.Now.ToLocalTime(), message);
    }
}
```

- 클래스를 사용한 인스턴스화 => 참조형식
I인터페이스명 인스턴스명 = new 클래스명();

```
ILogger logger = new ConsoleLogger();
logger.WriteLog( "Hello, World!" );
```

02. 인터페이스 선언 및 사용

// ICar 인터페이스 정의

참조 1개

```
interface ICar
{
    2
    void Go();
}
```

// ICar 인터페이스를 상속하는 Car클래스 선언

참조 2개

```
class Car : ICar
{
    참조 2개
    public void Go()
    {
        Console.WriteLine("Go");
    }
}
```

// 파생 클래스를 이용한 인스턴스화

```
Car car = new Car();
car.Go();
```

Go

02. 인터페이스 선언 및 사용

```
interface ICar2
{
    참조 4개
    void Work();
}
```

```
참조 2개
class Sedan : ICar2
{
    참조 2개
    public void Work()
    {
        Console.WriteLine("승용차가 사람을 태우고 있습니다.");
    }
}
```

02. 인터페이스 선언 및 사용

```
class Truck : ICar2
{
    참조 2개
    public void Work()
    {
        Console.WriteLine("트럭이 짐을 싣고 있습니다.");
    }
}
```

```
Sedan sedan1 = new Sedan();
sedan1.Work();
```

```
Truck truck1 = new Truck();
truck1.Work();
```

승용차가 사람을 태우고 있습니다.
트럭이 짐을 싣고 있습니다.

퀴즈

- 인터페이스 IDino를 이용하여 Dino 클래스를 선언하고 객체화 시켜 결과를 확인하여라.

```
interface IDino
{
    참조 3개
    void Eat(string what);
    참조 2개
    void Sleep(string where);
    참조 2개
    void Play(int num);
    참조 2개
    void Today(string what, int num, string where);
}
```

```
고기 을/를 먹는다.
생선 을/를 먹는다.
동굴 에서 잔다
친구 3 명과 놀고 있다.
===== 티라노 티티의 하루 일상 =====
사과 을/를 먹는다.
친구 2 명과 놀고 있다.
숲속 에서 잔다
```

퀴즈

```
class Tyrano:IDino
```

```
{
```

코드 작성

```
}
```

퀴즈

```
Tyrano tyrano = new Tyrano();
```

코드 작성

03. 인터페이스의 상속

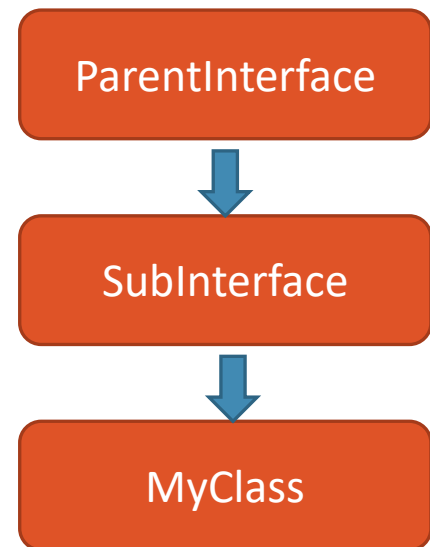
- 기존 인터페이스에 새로운 기능을 추가한 인터페이스를 만들고 싶을 경우 인터페이스를 상속한다.
- 실제 인터페이스를 상속받는 클래스에서 추상 메서드를 모두 구현해야 한다.
- 인터페이스의 상속 선언

```
interface 파생(자식)인터페이스 : 기반(부모) 인터페이스
{
    // 명령어
}
```


03. 인터페이스의 상속

```
interface ParentInterface
{
    참조 1개
    void myMethod(string str);
}

참조 1개
interface SubInterface : ParentInterface
{
    참조 1개
    void myMethod(string str, int i);
}
```



03. 인터페이스의 상속

```
class MyClass : SubInterface
{
    참조 1개
    public void myMethod(string str)
    {
        Console.WriteLine(str + " ParentInterface.myMethod() call!");
    }

    참조 1개
    public void myMethod(string str, int count)
    {
        for (int i = 0; i < count; i++)
        {
            Console.WriteLine(str + " SubInterface.myMethod() " + i + " call!");
        }
    }
}
```

03. 인터페이스의 상속

```
Console.WriteLine("인터페이스 상속 테스트");  
Console.WriteLine("\n-----");  
  
MyClass myclass = new MyClass();  
  
myclass.myMethod("Interface");  
Console.WriteLine();  
myclass.myMethod("Inherits", 4);
```

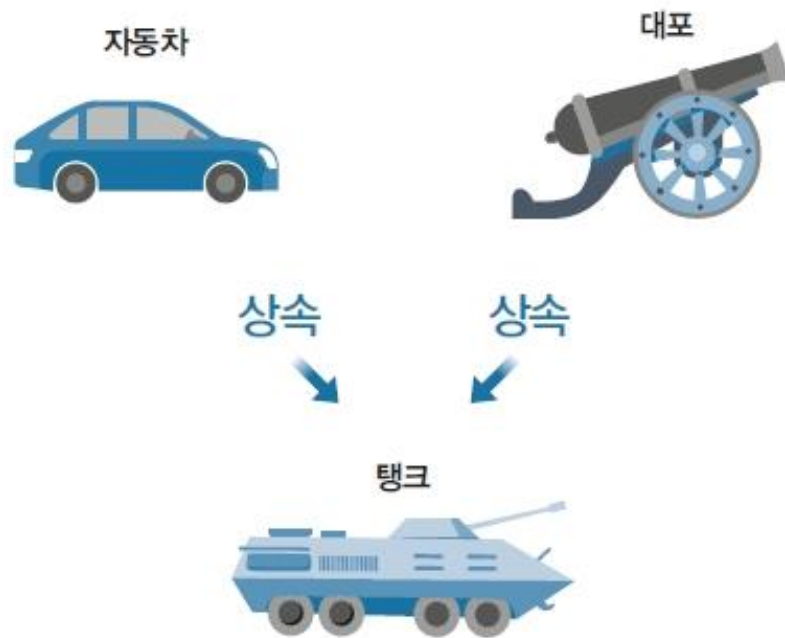
```
Interface ParentInterface.myMethod() call!  
  
Inherits SubInterface.myMethod() 0 call!  
Inherits SubInterface.myMethod() 1 call!  
Inherits SubInterface.myMethod() 2 call!  
Inherits SubInterface.myMethod() 3 call!
```

04. 인터페이스 다중 상속

- C#의 클래스에서는 다중 상속이 불가능하지만 인터페이스에서는 콤마(,)를 이용하여 다중 상속 인터페이스가 가능하다.
- 인터페이스 다중 상속 선언

```
interface 파생(자식)인터페이스 : 기반(부모) 인터페이스1, 기반(부모) 인터페이스2...  
{  
    // 추가할 메소드 목록  
}
```

04. 인터페이스 다중 상속



탱크 = 자동차 + 대포

04. 인터페이스 다중 상속

```
interface IRunnable
```

```
{  
    참조 2개  
    void Run();  
}
```

참조 1개

```
interface IFlyable
```

```
{  
    참조 2개  
    void Fly();  
}
```

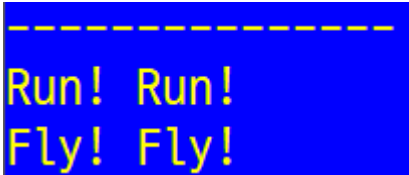
참조 2개

```
class FlyingCar : IRunnable, IFlyable
```

```
{  
    참조 2개  
    public void Run()  
    {  
        Console.WriteLine("Run! Run!");  
    }  
  
    참조 2개  
    public void Fly()  
    {  
        Console.WriteLine("Fly! Fly!");  
    }  
}
```

04. 인터페이스 다중 상속

```
FlyingCar flyingcar = new FlyingCar();  
flyingcar.Run();  
flyingcar.Fly();
```



```
-----  
Run! Run!  
Fly! Fly!
```

04. 인터페이스 다중 상속

```
interface ICar3
{
    참조 2개
    void Run();
}
```

```
참조 1개
interface ICannon3
{
    참조 2개
    void Fire();
}
```

```
참조 2개
class Tank : ICar3, ICannon3
{
    참조 2개
    public void Run()
    {
        Console.WriteLine("탱크가 앞으로 굴러갑니다.");
    }

    참조 2개
    public void Fire()
    {
        Console.WriteLine("탱크에서 대포를 발사합니다.");
    }
}
```

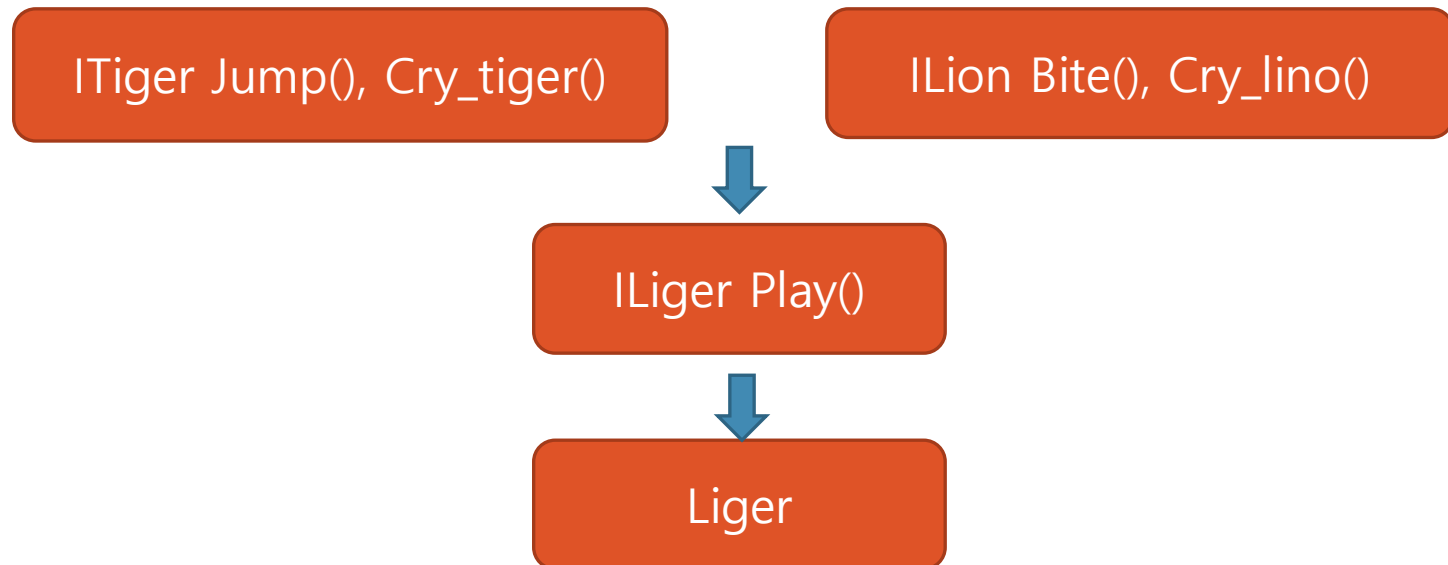

04. 인터페이스 다중 상속

```
Tank tank = new Tank();  
tank.Run();  
tank.Fire();
```

탱크가 앞으로 굴러갑니다.
탱크에서 대포를 발사합니다.

퀴즈

- 인터페이스 ITiger와 ILion을 상속받는 ILiger를 선언한 후 ILiger를 상속받는 Liger 클래스를 작성하여라. 정의된 추상 메서드는 아래 그림을 참조한다.



퀴즈

```
interface ITiger
```

```
{
```

코드 작성

```
}
```

참조 1개

```
interface ILion
```

```
{
```

코드 작성

```
}
```

참조 1개

```
interface ILiger:ITiger, ILion
```

```
{
```

코드 작성

```
}
```

참조 3개

```
class Liger:ILiger
```

```
{
```

코드 작성

```
}
```

퀴즈

```
Liger liger = new Liger("꿈돌이");  
Console.WriteLine($"\\t 라이거 {liger.Name} 의 하루 ");  
liger.Cry_tiger();  
liger.Play("사육사");  
liger.Bite();  
liger.Cry_tiger();  
liger.Cry_lion();  
liger.Jump();
```

라이거 꿈돌이 의 하루
호랑이처럼 어흥 ~~~
사육사와 놀기
사자처럼 한입에 꿀꺽하기
호랑이처럼 어흥 ~~~
사자처럼 으르렁 ~~~
호랑이처럼 점프하기

05. 클래스 VS 추상클래스 VS 인터페이스

