

컴퓨터정보과 C# 프로그래밍

9주차 상속과 다형성

강의 순서

1. C# 환경설치 / C# 기본 구조
2. 클래스 기본(필드) + 변수, 자료형
3. 클래스 기본(메소드) + 연산자, 수식
4. 클래스 기본(메소드) + 제어문
5. 배열/리스트/딕셔너리
6. 클래스 기본: 접근제한자 (한정자) + 프로퍼티(속성)
7. **클래스 심화: 상속**
8. 클래스 심화: 인터페이스/추상 클래스
9. 예외처리/일반화
10. 파일처리
11. UI (Winform or WPF)
12. LINQ/Delegate/Lambda/...

복습 및 7주차 추가요소

- 생성자와 this
- DateTime & TimeSpan
- 접근제한자의 보호수준
 - private > protected > internal > public

상속 (Inheritance)

코드이 재사용과 확장

상속 (Inheritance)

- 상속은 특정 기능(데이터 및 동작)을 제공하는 기본 클래스를 정의하고 해당 기능을 상속하거나 재정의하는 파생 클래스를 정의할 수 있는 객체 지향 프로그래밍 언어의 기능
- 상속은 객체 지향 프로그래밍의 기본적인 특성 중 하나입니다. 부모 클래스의 동작을 다시 사용(상속), 확장 또는 수정하는 자식 클래스를 정의할 수 있습니다. 멤버가 상속되는 클래스를 기본 클래스(base class)라고 합니다. 기본 클래스의 멤버를 상속하는 클래스를 파생 클래스(derived class)라고 함.
- 코드 재활용
 - 상속을 사용하면 다른 클래스에 정의된 동작을 다시 사용, 확장 및 수정하는 새 클래스를 만들 수 있습니다
- 단일 상속만 지원
 - 다중 상속 지원 언어 : C++, Python
- 생성자/종료자/private 멤버는 상속되지 않음.

상속 기본 문법

```
class 기본클래스  
{  
  
}
```

```
class 파생클래스 : 기본클래스  
{
```

//기본 클래스에서 선언한 멤버(필드, 메소드, 프로퍼티, ...)를 물려받는다.

//단, private 멤버는 물려 받을 수 없다.

```
}
```

기본클래스 - 부모클래스
파생클래스 - 자식클래스

예제

```
class Top (1)
{
    public int Age;
    public override string ToString()
    {
        return Age.ToString();
    }
}

class Bottom
{
    public int Age;
    public override string ToString()
    {
        return Age.ToString();
    }
}
```

```
class Top (2)
{
    public int Age;
    public override string ToString()
    {
        return Age.ToString();
    }
}

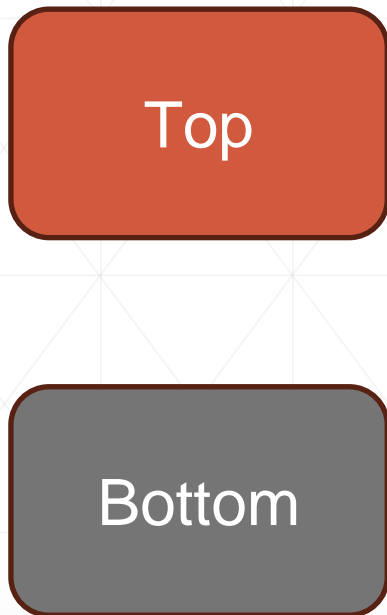
class Bottom : Top
{
    // Top의 멤버를 모두 사용할 수 있다.
}
```

```
Top top = new Top();
Bottom bottom = new Bottom();
top.Age = 20;
bottom.Age = 21;
```

```
Console.WriteLine(top);
Console.WriteLine(bottom);
```

표현

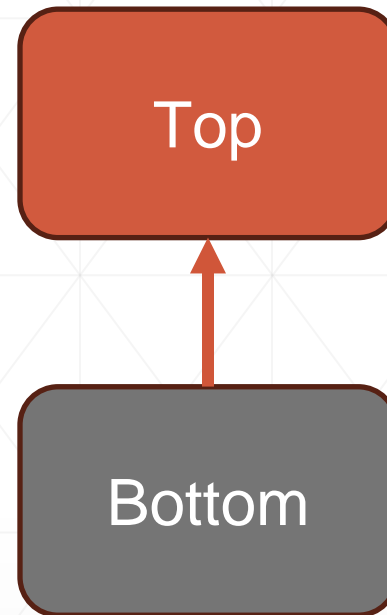
(1)



서로 관련 없는 class

Top class에 멤버가 추가되어도
Bottom class는 아무런 영향을 받지 않는다.

(2)

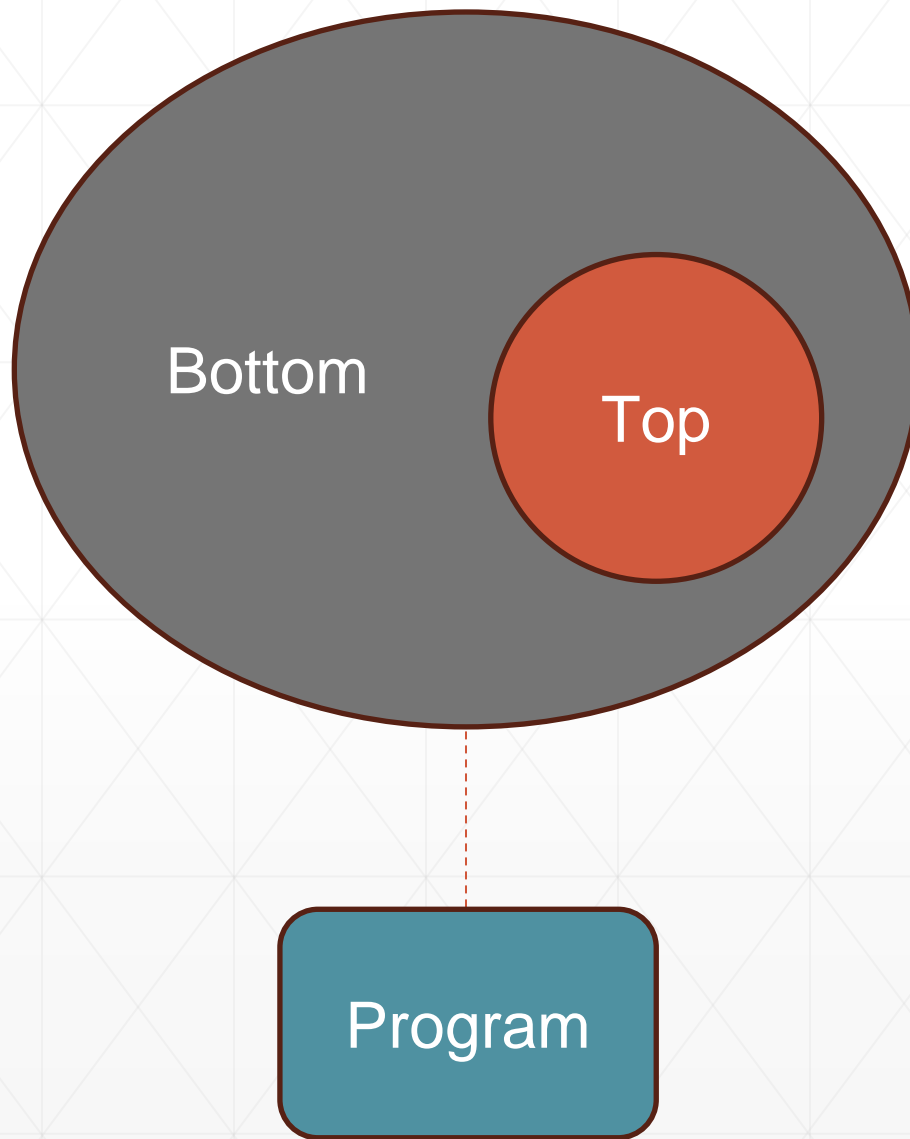
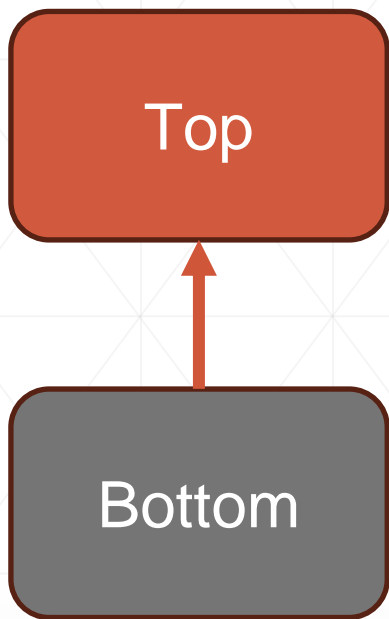


Top class는 Bottom class의 base class이다.
Bottom class는 Top class의 derived class이다.

Top class에 멤버가 추가되면
Bottom class도 동일하게 멤버가 추가된다.

표현

(2)



생성/종료의 관점

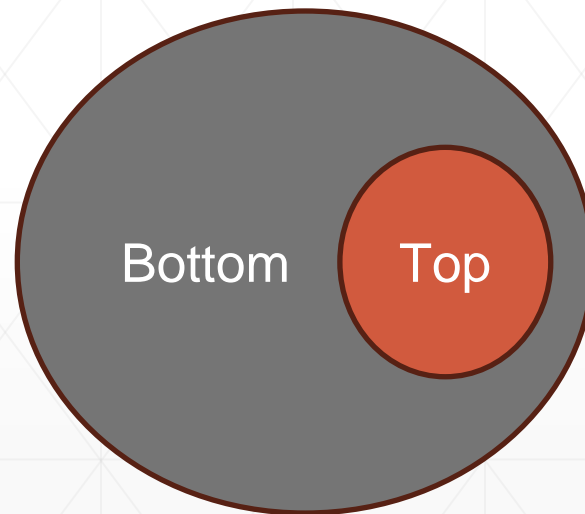
- 파생(derived) 클래스는 객체를 생성하는 시점에 먼저 기본(base) 클래스의 생성자를 호출한 후에 자신의 생성자를 호출한다.
- 파생(derived) 클래스의 객체 소멸은 파생(derived) 클래스의 종료자를 호출한 후에 기본(base) 클래스의 종료자를 호출한다.

```
class Top
{
    public Top() {
        Console.WriteLine("Top 생성자");
    }

    ~Top() {
        Console.WriteLine("Top 종료자");
    }
}

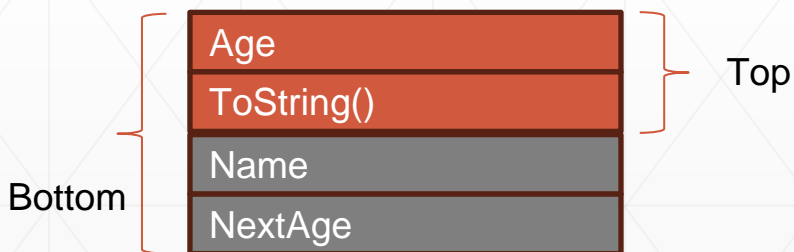
class Bottom : Top
{
    public Bottom() {
        Console.WriteLine("Bottom 생성자");
    }

    ~Bottom() {
        Console.WriteLine("Bottom 종료자");
    }
}
```



확장

- 기본클래스 + α \rightarrow 파생클래스
- 키워드
 - `this` : 현재 클래스의 객체
 - `base` : 기본 클래스의 객체
 - 이 키워드를 통해 기본 클래스에 접근할 수 있음.



```
class Top
{
    public int Age;
    public override string ToString()
    {
        return Age.ToString();
    }
}

class Bottom : Top
{
    public string Name;

    public int NextAge
    {
        get { return base.Age + 1; }
        //get { return this.Age + 1; }
        //get { return Age + 1; }
    }
}
```

base 생성자 (1)

```
class Top
{
    public int Age;

    public Top()
    {
    }
}

class Bottom : Top
{
    public string Name;

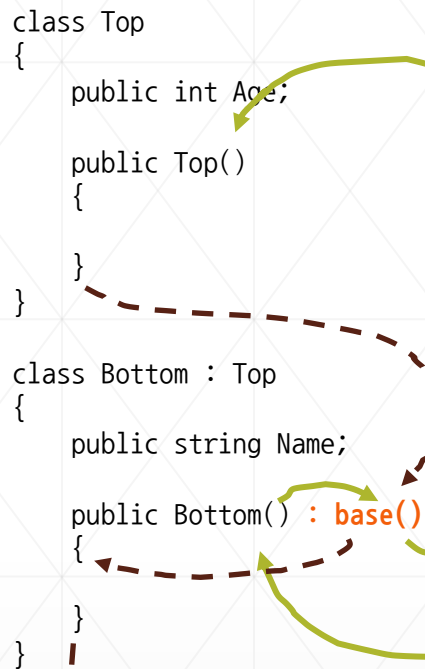
    public Bottom()
    {
    }
}
```

```
class Top
{
    public int Age;

    public Top()
    {
    }
}

class Bottom : Top
{
    public string Name;

    public Bottom() : base()
    {
    }
}
```



```
Bottom top = new Top();
Bottom btm = new Bottom();
```

base 생성자 (2)

```
class Top
{
    public int Age;

    public Top(int age)
    {
        Age = age;
    }
}

class Bottom : Top
{
    public string Name;

    public Bottom() ÷ base()
    {
    }
}
```

```
class Top
{
    public int Age;

    public Top(int age)
    {
        Age = age;
    }
}

class Bottom : Top
{
    public string Name;

    public Bottom() : base(0)
    {
    }
}
```

```
Bottom top = new Top(20);
Bottom btm = new Bottom()
```

base 생성자 (3)

```
class Top
{
    public int Age;

    public Top(int age)
    {
        Age = age;
    }
}

class Bottom : Top
{
    public string Name;

    public Bottom() ÷ base()
    {
    }
}
```

```
class Top
{
    public int Age;

    public Top(int age)
    {
        Age = age;
    }
}

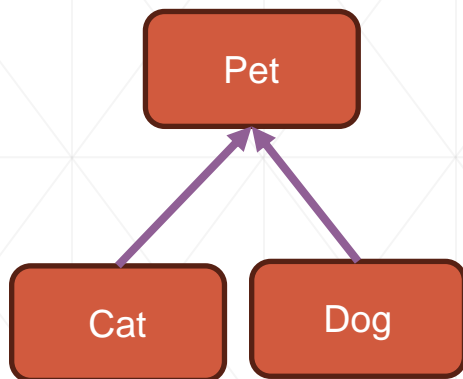
class Bottom : Top
{
    public string Name;

    public Bottom(int age, string name) : base(age)
    {
        Name = name;
    }
}
```

```
Bottom top = new Top(20);

Bottom btm = new Bottom(21, "김바닥");
```

() 연산자



```
class Pet
{
    public string Name;
    public int Age;
    public void Sleep()
    {
        Console.WriteLine("Zzzz");
    }
}

class Cat : Pet
{
    public void Meow()
    {
        Console.WriteLine("냐옹");
    }
}

class Dog : Pet
{
    public void Bark()
    {
        Console.WriteLine("멍멍");
    }
}
```

```
internal class Program
{
    static void Main(string[] args)
    {
        Pet pet = new Pet();
        Cat cat = new Cat();
        Dog dog = new Dog();

        var p1 = (Pet)pet;
        var p2 = (Pet)dog;
        var p3 = (Pet)cat;

        var c1 = (Cat)pet;
        var c2 = (Cat)dog;
        var c3 = (Cat)cat;

        var d1 = (Dog)pet;
        var d2 = (Dog)dog;
        var d3 = (Dog)cat;
    }
}
```

is 연산자

```
class Pet
{
    public string Name;
    public int Age;
    public void Sleep()
    {
        Console.WriteLine("Zzzz");
    }
}

class Cat : Pet
{
    public void Meow()
    {
        Console.WriteLine("냐옹");
    }
}

class Dog : Pet
{
    public void Bark()
    {
        Console.WriteLine("멍멍");
    }
}
```

```
internal class Program
{
    static void Main(string[] args)
    {
        Pet pet = new Pet();
        Cat cat = new Cat();
        Dog dog = new Dog();

        bool[] isType = new bool[9];

        isType[0] = pet is Pet ;
        isType[1] = dog is Pet;
        isType[2] = cat is Pet;

        isType[3] = pet is Cat;
        isType[4] = dog is Cat;
        isType[5] = cat is Cat;

        isType[6] = pet is Dog;
        isType[7] = dog is Dog;
        isType[8] = cat is Dog;

        for(int i=0; i < isType.Length; i++) {
            Console.WriteLine($"[{i}] {isType[i]}");
        }
    }
}
```


as 연산자

```
class Pet
{
    public string Name;
    public int Age;
    public void Sleep()
    {
        Console.WriteLine("Zzzz");
    }
}

class Cat : Pet
{
    public void Meow()
    {
        Console.WriteLine("냐옹");
    }
}

class Dog : Pet
{
    public void Bark()
    {
        Console.WriteLine("멍멍");
    }
}
```

```
internal class Program
{
    static void Main(string[] args)
    {
        Pet pet = new Pet();
        Pet cat = new Cat();
        Pet dog = new Dog();

        Pet p;
        if(dog is Pet) {
            p = (Pet)dog;
            Console.WriteLine("dog is Pet");
            Console.WriteLine(p);
        }

        Dog d;
        d = dog as Dog;
        if (d != null) {
            Console.WriteLine("dog as Dog");
            Console.WriteLine(d);
        }

        Cat c;
        c = dog as Cat;
        if (c != null) {
            Console.WriteLine("dog as Cat");
            Console.WriteLine(c);
        }
    }
}
```

향상된 is 연산자

```
class Pet
{
    public string Name;
    public int Age;
    public void Sleep()
    {
        Console.WriteLine("Zzzz");
    }
}

class Cat : Pet
{
    public void Meow()
    {
        Console.WriteLine("냐옹");
    }
}

class Dog : Pet
{
    public void Bark()
    {
        Console.WriteLine("멍멍");
    }
}
```

```
internal class Program
{
    static void Main(string[] args)
    {
        Pet pet = new Pet();
        Pet cat = new Cat();
        Pet dog = new Dog();

        if(dog is Pet p) {
            Console.WriteLine("dog is Pet");
            Console.WriteLine(p);
        }

        if (dog is Dog d) {
            Console.WriteLine("dog as Dog");
            Console.WriteLine(d);
        }

        if (dog is Cat c) {
            Console.WriteLine("dog as Cat");
            Console.WriteLine(c);
        }
    }
}
```

protected 접근제한자 (1)

```
class A
{
    private int _a;
    protected int _b;
    public int C;
    public void PrintA()
    {
        Console.WriteLine(this._a);
        Console.WriteLine(this._b);
        Console.WriteLine(this.C);
    }
}

class B : A
{
    public void PrintA()
    {
        Console.WriteLine(base._a);
        Console.WriteLine(base._b);
        Console.WriteLine(base.C);
    }
}

class C
{
    public void PrintC()
    {
        B b = new B();
        Console.WriteLine(b._a);
        Console.WriteLine(b._b);
        Console.WriteLine(b.C);
    }
}
```

**base 대신 this를 써도 무방하다.*

- protected
 - derived class는 접근 가능하지만 외부 class는 접근이 불가능한.

protected 접근제한자 (2)

```
class Pet
{
    private string _name;
    protected int _age;
    protected string Name { get { return _name; } }
    public int Age { get { return _age; } }

    public Pet(string name, int age)
    {
        _name = name;
        _age = age;
    }
}

class Cat : Pet
{
    public Cat(string name, int age) : base(name, age)
    {
    }

    public void Meow()
    {
        //Console.WriteLine($"[{_name}] [{_age}]냐옹");
        Console.WriteLine($"[{Name}] [{_age}]냐옹");
    }
}
```

```
class Dog : Pet
{
    public Dog(string name, int age) : base(name, age)
    {
    }

    public void Bark()
    {
        //Console.WriteLine($"[{_name}] [{_age}]멍멍");
        Console.WriteLine($"[{Name}] [{_age}]멍멍");
    }
}

internal class Program
{
    static void Main(string[] args)
    {
        Pet pet = new Pet("김펫", 2);
        Pet cat = new Cat("김냥", 3);
        Pet dog = new Dog("김멍", 5);

        cat.Meow();
        ((Cat)cat).Meow();
        Console.WriteLine(cat.Name);
        Console.WriteLine(cat.Age);

        dog.Bark();
        ((Dog)dog).Bark();
        Console.WriteLine(dog.Name);
        Console.WriteLine(dog.Age);
    }
}
```

다형성 (polymorphism)

코드이 재사용과 확장

다형성, Polymorphism

- 다형성이란 프로그램 언어 각 요소들(상수, 변수, 식, 객체, 메소드 등)이 다양한 자료형(type)에 속하는 것이 허가되는 성질을 가리킨다.
 - 런타임에 파생 클래스의 객체가 메서드 매개 변수 및 컬렉션 또는 배열과 같은 위치에서 기본 클래스의 객체로 처리될 수 있습니다. 이러한 다형성이 발생하면 객체의 선언된 형식이 더 이상 해당 런타임 형식과 같지 않습니다.
 - 기본 클래스는 가상메서드를 정의 및 구현할 수 있으며, 파생 클래스는 이러한 가상 메서드를 재정의할 수 있습니다. 즉, 파생 클래스는 고유한 정의 및 구현을 제공합니다. 런타임에 클라이언트 코드에서 메서드를 호출하면 CLR은 객체의 런타임 형식을 조회하고 가상 메서드의 해당 재정의 호출합니다. 소스 코드에서 기본 클래스에 대해 메서드를 호출하여 메서드의 파생 클래스 버전이 실행되도록 할 수 있습니다.

오버라이딩과 하이딩 (1)

- 가상 메소드는 디자이너에게 파생 클래스의 동작에 대한 다양한 선택권을 제공
 - 파생 클래스는 재정의하지 않고 가장 가까운 기본 클래스 메서드를 상속할 수 있어 기존 동작을 유지하지만 추가 파생 클래스가 메서드를 재정의할 수 있습니다.
 - 파생 클래스가 기본 클래스의 가상 메소드를 재정의하여 새로운 동작을 정의할 수 있습니다. (method overriding)
 - 파생 클래스가 기본 클래스 구현을 숨기는 멤버의 새로운 비가상 구현을 정의할 수 있습니다. (method hiding)
- 파생(derived) 클래스는 기본(base) 클래스 멤버가 **virtual** 또는 **abstract**로 선언된 경우에만 기본 클래스 멤버를 재정의(override)할 수 있습니다. 파생 멤버는 **override** 키워드를 사용하여 메서드가 가상 호출에 참여하도록 되어 있음을 명시적으로 나타내야 합니다.
- 파생(derived) 클래스가 기본(base) 클래스의 멤버와 동일한 이름을 갖는 멤버를 갖도록 하려면 **new** 키워드를 사용하여 기본 클래스 멤버를 숨길 수(hiding) 있습니다.
- 재정의한 파생 클래스는 계속해서 **base** 키워드를 사용하여 기본 클래스에 대한 메서드 또는 속성에 액세스할 수 있습니다. 다음 코드는 예제를 제공합니다.

오버라이딩과 하이딩 (2)

```

class Pet {
    public void Sleep() //일반 메소드
    {
        Console.WriteLine("Zzzz");
    }

    public virtual void Eat() //가상 메소드
    {
        Console.WriteLine("우걱우걱");
    }
}

class Cat : Pet {
    public new void Sleep() //하이딩 가능
    {
        Console.WriteLine("고양이가 잔다 골골");
    }

    public override void Eat() //오버라이딩 가능
    {
        Console.WriteLine("고양이가 먹는 뽐뽐");
    }
}

class Dog : Pet {
    public new void Sleep() //하이딩만 가능
    {
        Console.WriteLine("멍멍이가 잔다 드르렁");
    }

    public new void Eat() //하이딩도 가능
    {
        Console.WriteLine("멍멍이가 먹는 남남");
    }
}
  
```

```

internal class Program
{
    static void Main(string[] args)
    {
        Pet cat = new Cat();
        Pet dog = new Dog();

        Console.WriteLine("-----");
        cat.Sleep();
        cat.Eat();
        Console.WriteLine("-----");
        dog.Sleep();
        dog.Eat();
        Console.WriteLine("-----");
        ((Cat)cat).Sleep();
        ((Cat)cat).Eat();
        Console.WriteLine("-----");
        ((Dog)dog).Sleep();
        ((Dog)dog).Eat();
    }
}
  
```

```

Zzzz
고양이가 먹는 뽐뽐
-----
Zzzz
우걱우걱
-----
고양이가 잔다 골골
고양이가 먹는 뽐뽐
-----
멍멍이가 잔다 드르렁
멍멍이가 먹는 남남
  
```