클래스-1

이것이 C#이다

Contents

- 객체 지향 프로그래밍과 클래스
- 클래스의 선언과 객체의 생성
- 객체의 삶과 죽음에 대하여: 생성자와 종료자
- 정적 필드와 메소드
- 객체 복사하기: 얕은 복사와 깊은 복사
- this 키워드
- 접근 한정자로 공개 수준 결정하기
- 상속으로 코드 재활용하기
- 기반 클래스와 파생 클래스 사이의 형식 변환, 그리고 is와 as

7.1 객체 지향 프로그래밍과 클래스

- **코드 내의 모든 것을 객체로 표현하려는 프로그래밍 패러다임**
  - **객체**(Object) – **세상의 모든 것을 지칭**
- **객체의 표현**
  - **속성** - **데이터**
  - **기능** - **메소드**
- **클래스 – 객체를 만들기 위한 청사진**
- **int a = 30;**
  - int: **클래스**, **청사진**
  - a: **객체**, int**의 실체**(instance)

7.2 클래스의 선언과 객체의 생성

❖ 클래스의 선언 형식

```
class 클래스이름
{
    // 데이터와 메소드
}
```

❖ 클래스의 선언과 사용 예

```
class Cat
{
    public string Name;
    public string Color;

    public void Meow()
    {
        Console.WriteLine("{0} : 야옹", Name);
    }
}
```

데이터

메소드

```
Cat kitty = new Cat();
kitty.Color = "하얀색";
kitty.Name = "키티";
kitty.Meow();
Console.WriteLine("{0} : {1}", kitty.Name, kitty.Color);

Cat nero = new Cat();
nero.Color = "검은색";
nero.Name = "네로";
nero.Meow();
Console.WriteLine("{0} : {1}", nero.Name, nero.Color);
```

kitty 객체 생성

nero 객체 생성

❖ 데모 예제 - BasicClass

7.3 객체의 삶과 죽음에 대하여: 생성자와 종료자

- 객체의 삶과 죽음을 관장하는 두 가지 메소드
- 객체를 만드는 생성자
- 객체를 파괴하는 종료자

7.3.1 생성자

- 클래스와 같은 이름, 반환 형식 없음
- 선언 형식

```
class 클래스이름
{
    한정자 클래스이름( 매개변수목록 )
    {
        //
    }

    // 필드
    // 메소드
}
```

생성자

- 기본 생성자
- 사용자 지정 생성자

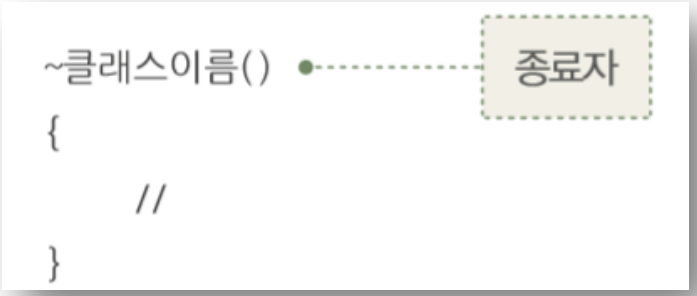
```
class Cat
{
    public Cat()
    {
        Name = "";
        Color = "";
    }

    public Cat( string _Name, string _Color )
    {
        Name = _Name;
        Color = _Color;
    }
```

객체를 생성할 때 이름과 색을 입력받아 초기화합니다.

```
Cat kitty = new Cat();    // Cat()
kitty.Name = "키티";
kitty.Color = "하얀색";

Cat nabi = new Cat( "나비", "갈색" ); // Cat( string _Name, string _Color )
```

7.3.2 종료자

- 클래스 이름 앞에 ~를 붙인 꼴
- 특징
  - 매개 변수도 없고, 한정자도 사용하지 않음
  - 오버로딩 불가능, 직접 호출할 수 없음
  - → CLR의 가비지 컬렉터가 객체 소멸 시점을 판단해서 종료자 호출
- 선언 형식



- 종료자는 사용하지 말자
  - CLR의 가비지 컬렉터의 동작 시점 예측 불가능
  - 명시적 종료자 구현은 성능 저하 초래 가능성 높음
  - CLR의 가비지 컬렉터는 객체 소멸 처리 전문가다.

7.4 정적 필드와 메소드

- static은 메소드나 필드가 클래스 자체에 소속되도록 지정하는 한정자
- 인스턴스 소속 필드 vs. 클래스 소속 필드

인스턴스에 소속된 필드의 경우

```
class MyClass
{
    public int a;
    public int b;
}

//

public static void Main()
{
    MyClass obj1 = new MyClass();
    obj1.a = 1;
    obj1.b = 2;

    MyClass obj2 = new MyClass();
```

클래스에 소속된 필드의 경우(static)

```
class MyClass
{
    public static int a;
    public static int b;
}

//

public static void Main()
{
    MyClass.a = 1;
    MyClass.b = 2;
}
```

인스턴스를 만들지 않고 클래스의 이름을 통해 필드에 직접 접근합니다.

- 프로그램 전체에 공유 하는 변수에 사용
- 데모 예제 - StaticField

7.4 정적 필드와 정적 메소드

❖ 정적 메소드는 클래스 자체에 소속됨

- 클래스 인스턴스 생성 없이도 호출 가능

❖ 선언 형식

```
class MyClass
{
    public static void StaticMethod()
    {
        // …
    }
```

```
class MyClass
{
    public void InctanceMethod()
    {
        // …
    }
}

// …
MyClass obj = new MyClass();
obj.InstanceMethod();
```

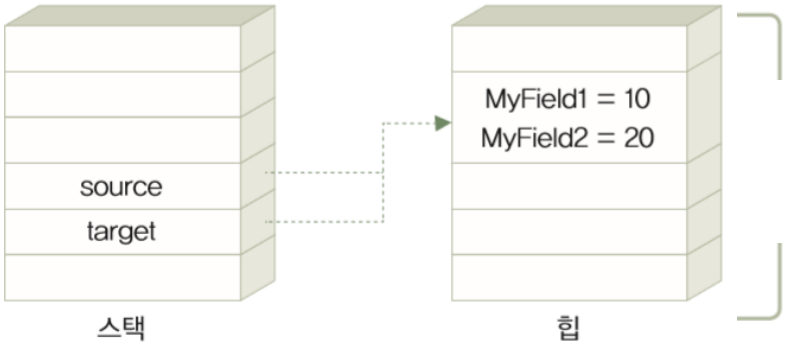
인스턴스를 만들어야 호출 가능

7.5 객체 복사하기: 얕은 복사와 깊은 복사

❖ 얕은 복사

- 객체를 복사할 때 참조만 살짝 복사

```
class MyClass
MyClass target = source;
target.MyField2 = 30;
}
```



❖ 깊은 복사

```
class MyClass
{
    public int MyField1;
    public int MyField2;
    public MyClass DeepCopy()
    {
        MyClass newCopy = new MyClass();
        newCopy.MyField1 = this.MyField1;
        newCopy.MyField2 = this.MyField2;
        return newCopy;
    }
}
```

객체를 힙에 새로 할당해서 그곳에 자신의 멤버를 일일이 복사해 넣습니다.

❖ 데모 예제 – DeepCopy

7.6 this

- 객체가 자신을 지칭할 때 사용하는 키워드 this
  - 객체 내부에서 자신의 필드나 메소드에 접근할 때 사용
- 사용 예

```
class Employee
{
    private string Name;

    public void SetName( string Name )
    {
        this.Name = Name;
    }
}
```

- this() 생성자
- 데모 예제
  - This
  - ThisConstructor

```
class MyClass
{
    int a, b, c;

    public MyClass()
    {
        this.a = 5425;
    }
    public MyClass(int b)
    {
        this.a = 5425;
        this.b = b;
    }
    public MyClass(int b, int c)
    {
        this.a = 5425;
        this.b = b;
        this.c = c;
    }
}
```

7.7 접근 한정자로 공개 수준 결정하기

❖ 은닉성(캡슐화)의 구현

- 감추고 싶은 것은 감추고, 보여주고 싶은 것만 보여준다.

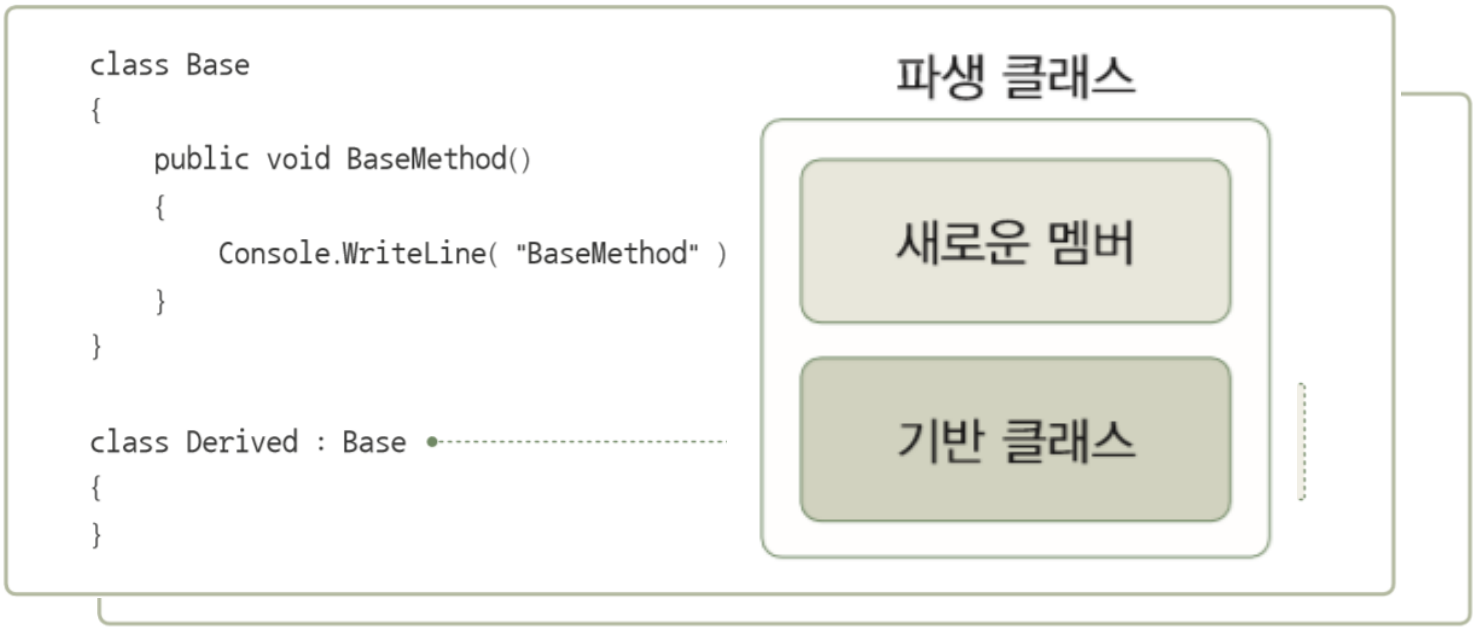
```
class MyClass
{
    private   int MyField_1;
    protected int MyField_2;
              int MyField_3;

    public int MyMethod_1( )
    {
        // …
    }

    internal void MyMethod_1 ( )
    {
        // …
    }
}
```

접근 한정자로 수식하지 않으면 private와 같은 공개 수준을 가집니다.

❖ 데모 예제 – AccessModifier

7.8 상속으로 코드 재활용하기(1)

❖ 물려받는 클래스가 물려줄 클래스 지정



❖ 파생 클래스 = 자신만의 고유 멤버 + 기반 클래스 멤버

7.8 상속으로 코드 재활용하기(2)

❖ 파생 클래스의 수명 주기

- 기반 생성자 → 파생 생성자 → 파생 종료자 → 기반 종료자

❖ 기반 클래스의 멤버 호출 → base

- 파생 클래스의 생성자에서 기반 클래스 생성자에 매개변수 전달

```
class Base
{
    protected string Name;
    public Base(string Name)
    {
        this.Name = Name;
    }
}

class Derived : Base
{
    public Derived(string Name) : base(Name)
    {
        Console.WriteLine("{0}.Derived()", this.Name);
    }
}
```

Base(string Name)을 호출

❖ 데모 예제 - Inheritance

7.9 기반 클래스와 파생 클래스 사이의 형식 변환

- 기반 클래스와 파생 클래스 사이에 족보를 오르내리는 형식 변환이 가능
- 파생 클래스의 인스턴스를 기반 클래스의 인스턴스로 사용 가능

```
Mammal mammal = new Mammal();
```

```
class Zookeeper
```

```
class Zookeeper
{
    public void Wash( Mammal mammal ) { /* … */ }
}
```

```
}
```

```
Cat cat = (Cat)mammal;
cat.Nurse();
cat.Meow();
```

7.9 is와 as

❖ C#의 형 변환 연산자

연산자	설명
is	객체가 해당 형식에 해당하는지를 검사하여 그 결과를 bool 값으로 반환합니다.
as	형식 변환 연산자와 같은 역할을 합니다. 다만 형변환 연산자가 변환에 실패하는 경우 예외를 던지는 반면에 as 연산자는 객체 참조를 null로 만든다는 것이 다릅니다.

❖ is와 as(참조 형식에만 사용) 연산자의 사용법

```
Mammal mammal2 = new Cat();

Cat cat = mammal2 as Cat;
if (cat != null)
{
    cat.Meow();
}
```

mammal2가 Cat 형식 변환에 실패했다면 cat은 null이 됩니다. 하지만 이 코드에서는 mammal2는 Cat 형식에 해당하므로 안전하게 형식 변환이 이루어집니다.

❖ 데모 예제 - TypeCasting

Thank You !

이것이 C#이다