C# 교실

CLASS 3
– OOP Part1

고주형
Thanks to 박소현
2019/05/08

오늘 할 것

- 객체지향(Object-Oriented)
- 구조체(Struct)
- 클래스(Class)
- 생성자(Constructor)/소멸자(Destructor)
- 정적멤버(Static Member)
- 네임스페이스(NameSpace)

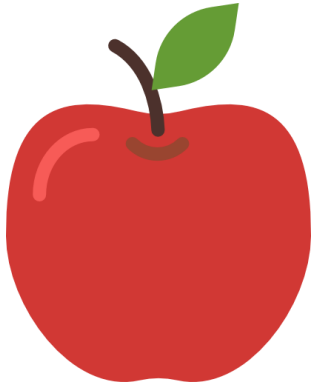
Introduction to OOP

#객체
#객체 지향
#객체지향프로그래밍(OOP)

객체(Object)가 무엇일까요?

책도

사람도



사과도

현실 세계에 있는 모든 것들이 객체!

객체(Object)가 무엇일까요?

어떤 **특징**을 만족을 해야 그 **객체**라고 부를 수 있을까?

ex. 책이라는 **객체**는 어떤 공통적인 **특징**을 가지고 있을까?

책

=

제목
ISBN 식별자
내용
저자
페이지수

속성(Attribute)

+

펼친다
덮는다
태운다
던진다
버린다

행위(Behavior)

위와 같은 방식으로 현실 세계의 모든 것을 정의하는 것을
객체 지향(Object Oriented)이라고 한다.

이를 프로그래밍에 적용하면?

class 책

=

string 제목
decimal ISBN
string 내용
string 저자
int 페이지수

필드(Field)

+

펼친다()
덮는다()
태운다()
던진다()
버린다()

메서드(Method)

객체지향 개념을 프로그래밍에 적용한 것을
객체지향 프로그래밍(Object-Oriented Programming)이라고 한다!

클래스

#구조체 #클래스vs구조체
#클래스vs인스턴스 #클래스
#필드 #메서드

구조체 정의

```
public struct Exam
{
        public string name;
        public int score;
        //생성자 *주의* 무조건 인자가 있어야합니다.
        public Exam(string _name,int _score)
        {
                this.name = _name;
                this.score = _score;
        }
        //메소드
        public void Print()
        {
                Console.WriteLine("이름: " + name + ", 성적: " + score);
        }
}
```

구조체 사용

```
static void Main(string[] args)
{
            //new 없이 사용
            Exam exam1;
            exam1.name = "그루트";
            exam1.score = 98;
            exam1.Print();
            //new 사용
            Exam exam2 = new Exam("고주형",90);
            exam2.Print();
}
```

구조체 vs 클래스

구조체	클래스
참조형	값형
new연산자 사용 안해도 됨	new연산자 사용해야 됨
인자가 없는 생성자 못 만듦	생성자가 인자가 없어도 됨
상속 못함	상속 가능
복사할 때 값 복사	복사할 때 참조 복사

Class vs. Instance

클래스

틀

ex. 붕어빵틀, 책

인스턴스/객체

틀로 찍어낸 것들

ex. 붕어빵,
책(걸리버 여행기)

클래스

- 클래스 정의

```
class Book
{
    // Book 클래스의 속성 정의
    string Title;
    decimal ISBM;
    string Contents;
    string Author;
    int PageCount;
}
```

클래스

- 클래스 사용

```
class Program
{
    static void Main(string[] args)
    {
        // Book타입의 인스턴스/객체 생성
        Book gulliver = new Book();
    }
}
```

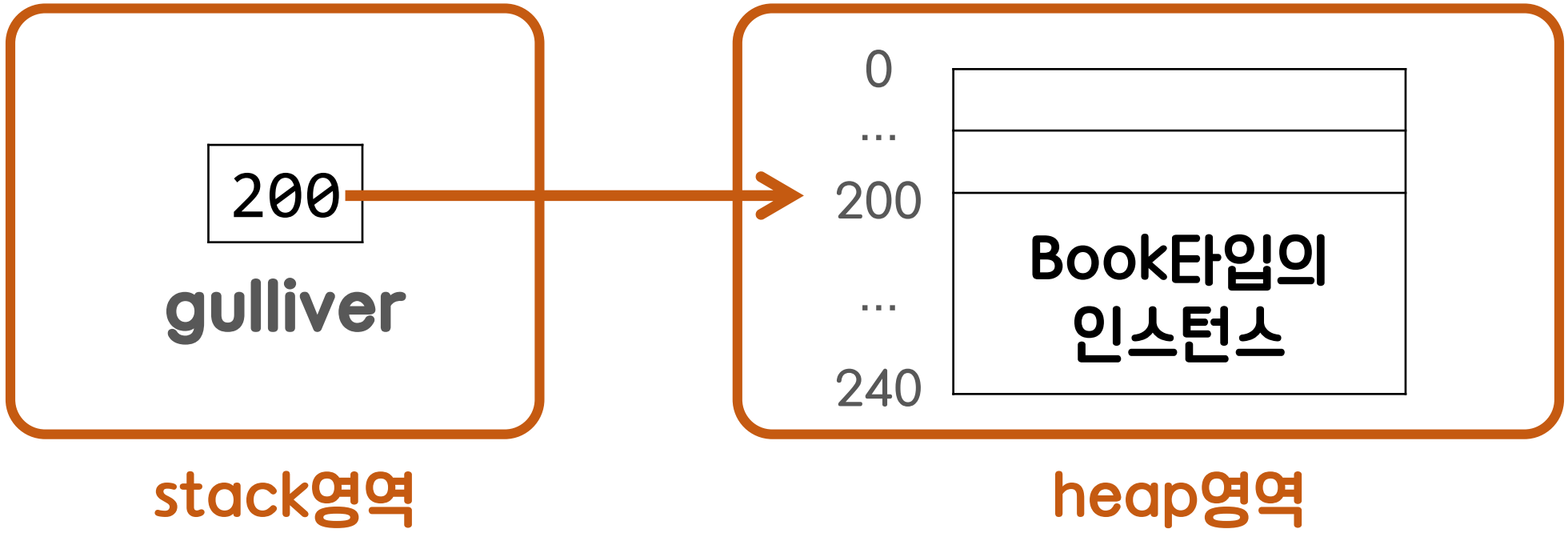
클래스는 내가 만든 자료형(Data Type)이다!

- 현재까지 배운 **기본 자료형(short, int, …)**으로만 우리 현실 세계의 복잡한 문제들을 해결하기에는 **역부족**이다.
- 기본 자료형은 사람이 아닌 **컴퓨터의 입장**에서 나타낸 자료형이므로, **표현의 한계**가 있다.
- 기본 자료형 외에 사람들이 **문제 해결**을 위해 **자신이 원하는 자료형**을 **새롭게 정의**하도록 한다.
- 이 자료형이 **클래스**이다!

클래스는 참조형 자료형이다!

- gulliver는 실제로 Book타입의 인스턴스가 저장되어 있는 메모리 주소를 담고 있다!

```
Book gulliver = new Book(); // 클래스의 크기는 40bytes
```



new 연산자

- 인스턴스/객체를 생성하는 연산자
- 클래스만 만든다고 인스턴스/객체가 생성되는 것이 아닙니다!

```
Book gulliver = new Book();
```

gulliver라는 변수가
오른쪽의 객체를 참조

Book타입의 객체를 생성

필드

- 클래스에서 속성(attributes)을 나타내는 변수들(variables)

클래스 필드 정의

```
class Book{
      public string Title;
      public decimal ISBN;
      public string Contents;
      public string Author;
      public int PageCount;
}
```

필드

- 클래스에서 속성(attributes)을 나타내는 변수들(variables)

클래스 필드 사용

```
class Program{
	static void Main(string[] args){
		Book gulliver = new Book();

		gulliver.Title = “걸리버 여행기”;
		gulliver.ISBN = 2947598294735m;
		gulliver.Contents = “내용임..애용“;
		gulliver.Author = “조나단 스위프트”;
		gulliver.PageCount = 364;

	}
}
```

메서드

- 클래스에서 행위(behaviors)를 나타내는 함수들(methods)

클래스 메서드 정의

```
class Book{
        …

        public void Open(){
                Console.WriteLine(“책 열었어요!”);
        }

        public void Close(){
                Console.WriteLine(“책 닫았어요!”);
        }
}
```

메서드

- 클래스에서 행위(behaviors)를 나타내는 함수들(methods)

클래스 메서드 사용

```
class Program{
	static void Main(string[] args){
		Book gulliver = new Book();

		gulliver.Open();
		gulliver.Close();
	}
}
```

책 열었어요!
책 닫았어요!

메서드는 왜 사용할까?

- 중복되는 코드를 제거하기 위해서
  (후에 유지/보수 때 중요)
- 코드 추상화를 위해

메서드는 왜 사용할까? – 중복되는 코드 제거

극단적으로 생각해봅시다!
당신은 개발자이고, 고객으로부터 다음과 같은 프로그램 주문 받았다고 합시다.

1부터 100까지의 숫자가

2로 나누어 떨어지는지

검사하는 프로그램을 만들어주세요!

```
static void Main(string[] args)
{
        int x = 1;
        if(x%2 == 0)
        {
                Console.Write(x);
        }

        x = 2;
        if(x%2 == 0)
        {
                Console.Write(x);
        }
        …
}
```

메서드는 왜 사용할까? – 중복되는 코드 제거

근데, 다 만들어놨더니 고객이 코드를 수정할 것을 요구합니다☹

100개나 되는 걸 언제 다 수정하지…

3으로 나누어 떨어지는지

검사하는 프로그램으로 수정해주세요!

```
static void Main(string[] args)
{
        int x = 1;
        if(x%2 == 0)
        {
                Console.Write(x);
        }

        x = 2;
        if(x%2 == 0)
        {
                Console.Write(x);
        }
        …
}
```

메서드는 왜 사용할까? – 중복되는 코드 제거

그래서 두 번 이상 중복되어 쓰이는 부분의 코드를 메서드로 만듭니다.
그러면 %2인 부분을 %3인 부분으로만 수정하면 되겠죠?

```
class Math
{
        public void PrintIf(int value)
        {
                if(value%3 == 0)
                {

        Console.WriteLine(value);
                }
        }
}
```

```
static void Main(string[] args)
{
        int x = 1;
        PrintIf(x);

        x = 2;
        PrintIf(x);

        …
}
```

메서드는 왜 사용할까?

- 중복되는 코드를 제거하기 위해서

  (후에 유지/보수 때 중요)

- 코드 추상화를 위해

메서드는 왜 사용할까? – 코드 추상화를 위해

한 번 고객의 입장이 되어봅시다!

방금 3으로 나누어지는 프로그램을 주문했는데, 고객이 어떻게 3으로 나누어지는지에 대한 과정을 알 필요가 있을까요?

아닙니다!

Printlf()가 3으로 나누어지게 하는 것이구나만 알면 됩니다!

메서드가 구체적으로 어떻게 구현되었는지 알 필요 없이
메서드의 용도만으로 메서드를 사용할 수 있습니다.

생성자/소멸자

#생성자
#소멸자 #GC(Garbage Collector)

생성자

- new연산자로 객체를 생성하는 시점에 자동으로 호출되는 특별한 메서드
  - 하지만, 반환타입이 없다.
- 생성자 이름은 클래스 이름과 같다.

```
class [클래스이름]
{
        [접근제한자] [클래스이름] ([타입] [매개변수명], … )
        {
        }
}
```

constructor를 줄여서 ctor라고 부른다.

생성자

- 기본 생성자(default constructor)
  - 매개변수가 없는 생성자
  - 명시적으로 어떤 생성자도 정의하지 않았다면, 컴파일러는 기본 생성자를 집어 넣고 컴파일한다.

```
class Book
{
	// 기본 생성자
	public Book()
	{
	}
}
```

생성자

- 매개변수를 갖는 생성자
  - 외부로부터 매개변수를 받을 수 있는 생성자
  - 주로 외부로부터 객체를 초기화하는 값을 입력받기 위해 사용된다.

```
class Book
{
        string Title;

        // 매개변수를 갖는 생성자
        Book(string title)
        {
                Title = title;
        }
}
```

생성자

- 명시적으로 기본생성자가 아닌 매개변수를 받는 생성자를 정의했을 경우, 컴파일러는 기본 생성자를 추가하지 않습니다.

```
class Book
{
	string Title;

	// 매개변수를 받는 생성자
	public Book(string
title)
	{
		Title = title;
	}
}
```

```
static void Main(string[] args)
{
	// 에러
	Book gulliver = new Book();
}

static void Main(string[] args)
{
	Book gulliver = new Book("걸리버 여행기");
}
```

생성자

- 생성자는 여러 개로 정의하는 것이 가능합니다.

```
class Book
{
        string Title;
        int PageCount;

        public Book()
        {
        }
        public Book(string title)
        {
                Title = title;
        }
        public Book(string title, int pageCount)
        {
                Title = title;
                PageCount = pageCount;
        }
}
```

소멸자

- 객체가 제거되는 시점에 실행되는 특별한 메서드
- C#에서는 객체를 제거하는 예약어(ex. delete)가 존재하지 않는다.
- 대신 Garbage Collector를 통해 사용이 끝난 객체를 자동으로 해제시키는 데 이 때 소멸자가 호출된다.

```
class 클래스이름
{
        ~클래스이름()
        {
                // 객체 해제를 위한 코드
        }
}
```

destructor를 줄여서 dtor라고 부른다.

정적 멤버

#정적멤버
#정적필드 #정적메서드 #정적생성자

정적멤버 vs 인스턴스 멤버

- 정적멤버
  - 각각의 인스턴스 수준이 아닌 해당 클래스 타입의 모든 인스턴스에 적용되는 필드, 메서드, 생성자

- 인스턴스멤버
  - 각각의 인스턴스와 관련된 필드, 메서드, 생성자

정적 필드

- 정적 필드 정의
  - 변수 선언 앞에 static 예약어를 붙여준다.

```
class Person
{
        // 정적필드 정의
        static public int Count = 0;
        // 인스턴스필드 정의
        public string Name;

        public Person(string name)
        {
                Name = name;
                Count++;
        }
}
```

정적 필드

- 정적 필드 사용
  - 사용자가 만든 인스턴스에서 만들어진 필드가 아니므로 클래스 이름으로 접근해주어야 한다.

    [클래스이름].[필드이름]

```
static void Main(string[] args)
{
        Console.WriteLine(Person.Count);

        Person person1 = new Person("고주형");
        Person person2 = new Person("박소현");

        Console.WriteLine(Person.Count);
}
```

출력
0
2

정적 필드

- 특정 클래스의 인스턴스를 의도적으로 딱 1개만 만들고 싶을 때 쓰인다.
  - 클래스 내부에 미리 인스턴스를 생성
  - 생성자를 private접근 제한자로 명시하여 외부에서 생성하지 못하게 막는다.

정적 필드

```
class Star
{
	// 클래스 내에서 인스턴스 미리 생성
	static public Star Sun = new Star(“태양”);

	string Name;

	// private으로 외부에서 객체가 생성되는 것을 막음
	private Star(string name)
	{
		Name = name;
	}

	// public 인스턴스 메서드
	public void DisplayStarName()
	{
		Console.WriteLine(Name);
	}
}
```

```
static void Main(string[] args)
{
	// 정적 필드로 하나만 존재하는 인스턴스에
	// 접근
	Star.Sun.DisplayStarName();

	// 오류
	// 생성자가 private이므로
	// 외부에서 객체를 생성할 수 없음
	Star Sun2 = new Star(“다른 태양”);
}
```

정적 메서드

- 정적 메서드 정의
  - 일반 메서드 앞에 static 예약어를 붙인다.
  - 정적 메서드 안에서는 인스턴스 멤버에 접근할 수 없다.
    - 즉, 정적 메서드 안에서는 정적 멤버에만 접근할 수 있다.

```
class Person
{
        …
        // public 정적 메서드
        static public void OutputCount()
        {
                // 정적 메서드에서 정적 필드에 접근
                Console.WriteLine(Count);
        }
}
```

정적 메서드

- 정적 메서드 사용
  - 사용자가 만든 인스턴스에서 만들어진 메서드가 아니므로 클래스 이름으로 접근해주어야 한다.

```
static void Main(string[] args)
{
        Person.OutputCount();

        Person person1 = new Person(“고주형”);
        Person person2 = new Person(“박소현”);

        Person.OutputCount();
}
```

출력
0
2

정적 메서드

- Main 메서드
  - static int Main(string[] args)

정적 생성자

- 정적 생성자 정의
  - 기본 생성자에 static예약어를 붙인다.
  - 1개만 정의할 수 있고 매개변수를 포함할 수 없다.

```
class 클래스이름
{
        static 클래스이름()
        {
                // 딱 한 번 가장 처음으로 실행될 초기화 코드
        }
}
```

static constructor를 줄여서 cctor라고 부른다.

정적 생성자

- 정적 생성자 사용
  - 주로 정적 멤버를 초기화하는 기능을 한다.
  - 클래스의 어떤 멤버든 최초로 접근하는 시점에 단 한 번 실행된다.

```
class Star
{
        static public Star Sun;
        public string Name;

        private Star(string name)
        {
                Name = name;
        }

        static Star()
        {
                Sun = new Star("태양");
        }
}
```

네임 스페이스

#네임스페이스
#using_예약어

네임 스페이스

- 이름이 중복되어 정의된 것을 구분하기 위해 사용된다.
- 일반적으로 수많은 클래스를 분류하는 방법으로 사용된다.

네임스페이스 선언

```
namespace Fruit
{
        class Apple
        {
        }
}

namespace Vegetable
{
        class Carrot
        {
        }
}
```

네임 스페이스

- 네임스페이스 안의 클래스를 사용하려면, 네임스페이스를 함께 지정해줘야 한다.
- 하지만, 이런 방식은 매우 번거롭다.

네임스페이스 사용

```
static void Main(string[] args)
{
        // 네임스페이스를 함께 지정해야 하기에 매우 번거롭다.
        Vegetable.Carrot carrot = new Vegetable.Carrot();
}
```

using 예약어

- 네임스페이스를 미리 선언해두어 객체를 생성시 네임스페이스를 생략해도 C# 컴파일러가 알아서 객체가 속한 네임스페이스를 찾아준다.
- using문은 반드시 파일 첫 부분에 있어야 한다.

```
using Vegetable;

namespace ConsoneApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
                // 네임스페이스를 함께 지정해야하기에 매우 번거롭다.
                Vegetable.Carrot carrot = new Vegetable.Carrot();
        }
    }
}
```

실습

출처

- [책] 시작하세요 C# 6.0