C# 객체지향프로그래밍

객체지향프로그래밍

- 객체지향프로그래밍(Object Oriented Programming)
  - 프로그래밍하고자 하는 대상을 속성(필드)과 행동(메소드)로 구성된 객체로 구현
  - 클래스
    - 객체를 만들기 위한 틀
    - 여러 속성들을 가지고있는 데이터 타입
    - 멤버 변수, 메소드, 생성자, 소멸자 등으로 구성
  - 객체
    - 메모리 상에 각각의 상태 값을 가지고 있는 클래스의 인스턴스

객체지향프로그래밍의 특성

- 은닉성
  - 클래스의 사용자에게 필요한 멤버 필드와 메소드만 공개
  - 클래스 외부에서 사용할 필요가 없는 멤버들은 숨긴다.
  - 코드의 보안을 위해 사용
- 상속성
  - 어떤 클래스의 내용을 다른 클래스에 전달하여 더 특화된 클래스를 만들기 수 있다.
  - 기반클래스(부모클래스)와 파생클래스(자식클래스)
  - 코드의 재사용
- 다형성
  - 한 행동을 여러방법으로 구현하고 상황에 따라 선택해서 쓸 수 있다.
    - 한 클래스에서 서로 다른 매개변수를 써서 하나의 메소드를 두가지 버전으로 정의
    - 기반클래스와 파생클래스의 같은 메소드를 다르게 정의

static 필드와 메소드

- **static**
  - 클래스에 정의된 필드나 메소드가 클래스의 인스턴스가 아닌 클래스 자체에 소속되도록 지정
  - 클래스의 인스턴스를 생성하지 않아도 사용 가능
  - 프로그램의 어디서든 사용할 경우 이용

this 키워드

- 객체 내부에서 자신이 가지고있는 필드나 메소드에 접근할 때 사용

```
Class MyClass
{
    private string name;

    public void setName(string name)
    {
        this.name = name;
    }
}
```

this 생성자

- 객체 자신의 생성자를 가리킴
- 중복되는 코드를 제거
- 생성자에서만 사용 가능

base 생성자

- 부모 클래스의 생성자를 가리킴
- 중복되는 코드를 제거
- 생성자에서만 사용 가능

접근 한정자

- 접근 한정자(Access Modifier)
  - 멤버 필드나 메소드의 접근 권한을 설정
  - 접근 한정자를 설정하지 않을 경우 자동으로 private로 설정됨

public	클래스 내/외부 모두 접근 가능
protected	클래스 외부에서는 접근할 수 없지만 파생 클래스에서는 접근 가능
private	클래스 내부에서만 접근 가능

객체의 복사

- 클래스의 객체가 생성되면 정의된 멤버 필드들의 값은 힙에 저장
- 클래스 인스턴스 변수는 멤버 필드의 데이터가 있는 주소 값을 참조함
- 얕은 복사(Shallow Copy)
  - 객체 복사 시 스택의 레퍼런스 값만 복사
- 깊은복사(Deep Copy)
  - 객체 복사 시 힙에 저장된 멤버 필드의 데이터까지 복사

클래스의 형식 변환

- 클래스 형식 변환
  - 기반 클래스와 파생 클래스 사이에는 형식 변환이 가능하다.
  - 코드의 생산성 향상
- is
  - 객체가 해당 형식에 해당하는지를 검사하여 그 결과를 bool 값으로 반환
- as
  - 형식 변환 연산자와 같은 역할. 형 변환 실패시 객체의 레퍼런스를 null로 만듬

오버라이딩

- 파생클래스의 메소드가 부모클래스가 정의한 메소드를 오버라이드

중첩 클래스

- 클래스안에 선언되어있는 클래스
- 자신이 소속되어있는 클래스의 멤버에 자유롭게 접근 가능(Private도 접근 가능)
- 클래스 외부에 공개하고 싶지 않은 형식을 정의할 때
- 현재 클래스의 일부분처럼 표현할 수 있는 클래스를 만들고자 할 때

분할 클래스

- 여러번에 나누어서 구현하는 클래스
- 소스코드 관리의 편의를 제공
- partial 키워드를 class 앞에 붙여줌