더 멋진 내일 Tomorrow 을 위한 내일 My Career

내일은 유니티 Unity

기초 입문편

이우령 지음

김앤북
KIM&BOOK

챕터 요약 정리

키워드로 정리하기

- **Unity**는 크로스 플랫폼 게임 개발을 지원하는 게임 엔진이다.
- Unity 하나만을 사용하여 게임을 만들 수 있지만 필요에 따라 **Visual Studio**와 같은 추가 도구를 사용할 수 있다.
- Unity는 **라이선스**를 등록해야 하고, 라이선스의 종류에 따라 Unity를 사용할 수 있는 용도가 달라진다.
- Unity의 주요 화면 구성으로 **Hierarchy**, **Inspector**, **Scene**, **Game** 등이 있다.

챕터 요약 정리

키워드로 정리하기

- Unity는 크게 **씬**, **오브젝트**, **컴포넌트**로 구성되고, 유니티를 이용한 게임 개발은 이들을 만들고 조립하는 과정이다.
- **씬**은 게임을 이루는 하나의 장면 또는 세상으로, 한 번에 하나의 씬이 실행된다.
- **오브젝트**는 씬에 포함되어 게임을 이루는 각각의 객체이다.
- **컴포넌트**는 오브젝트에 추가될 수 있는 기능, 능력, 역할을 나타낸다.
- 오브젝트는 부모 자식 관계를 맺을 수 있고, 자식 오브젝트는 부모 오브젝트의 위치, 크기, 회전에 영향을 받는다.

챕터 요약 정리

키워드로 정리하기

- Unity에서의 프로그래밍은 대부분 컴포넌트를 직접 만드는 것을 말한다.
- MonoBehaviour는 프로그래머가 컴포넌트를 만들기 위해 사용된다.
- **생명 주기**는 컴포넌트가 생성되고 종료될 때까지의 흐름을 나타낸다.
- 생명 주기 함수인 Start는 컴포넌트가 생성되었을 때, Update는 생성 이후 반복적으로 호출되는 함수다.
- Update는 컴퓨터마다 호출 횟수가 다르므로 Time.deltaTime을 통해 일관적인 결과를 만든다.
- 입력과 관련된 함수와 값은 Input, 시간과 관련된 함수와 값은 Time에 있다.
- 다른 컴포넌트에 접근하려면 GetComponent나 Inspector 창을 이용한다.

예제로 정리하기

손으로 익히는 코딩

```
public class MyComp : MonoBehaviour // ①
{
    Transform trans; // ②
    void Start() // ③
    {
        trans = GetComponent<Transform>(); // ④
    }
    void Update() // ⑤
    {
        trans.Translate(Time.deltaTime, 0, 0);
    }
}
```

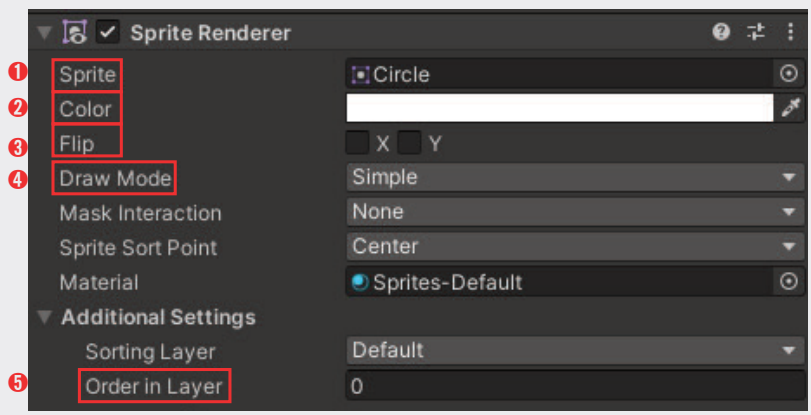
① 프로그래머가 직접 만드는 컴포넌트는 MonoBehaviour를 기본으로 한다.
② 컴포넌트에 추가된 멤버 변수는 컴포넌트가 삭제되기 전까지 유지된다.
③ Start는 컴포넌트가 시작될 때 호출되는 생명 주기 함수이다.
④ GetComponent는 오브젝트 또는 컴포넌트로부터 컴포넌트를 얻는다.
⑤ Update는 컴포넌트의 매 연산마다 호출되는 생명 주기 함수이다.

챕터 요약 정리

키워드로 정리하기

- **에셋**은 게임을 구성하는 다양한 요소이고, **패키지**는 에셋을 묶은 것이다.
- 에셋 및 패키지는 **유니티 에셋 스토어**, 공식 패키지 등으로 다운로드할 수 있다.
- 게임에서 사용되는 이미지를 **스프라이트**라고 한다.
- 스프라이트는 단일 스프라이트 또는 다중 스프라이트를 쓸 수 있으며, 다양한 설정값을 가진다.
- **스프라이트 렌더러**는 스프라이트를 렌더링하기 위한 렌더러이다.
- **라인렌더러**는 선을 렌더링하기 위한 렌더러이다.

이미지로 정리하기



스프라이트 렌더러는 오브젝트를 스프라이트로 렌더링하는 기능을 한다.

① Sprite : 화면에 보일 스프라이트이다.

② Color : 스프라이트에 적용할 색상이다. 흰색은 원본 스프라이트를 그대로 보이게 한다.

③ Flip : 좌우 반전, 상하 반전을 한다.

④ Draw Mode : 스프라이트를 연장하는 방식을 나타낸다.

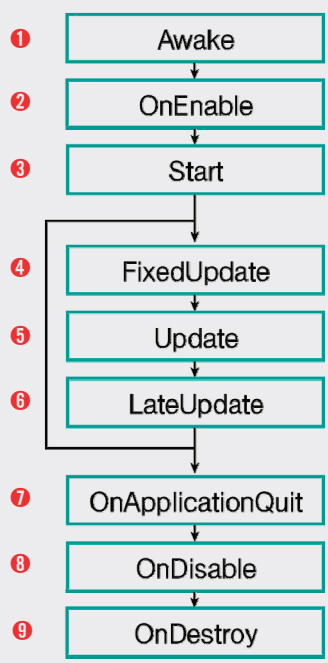
⑤ Order in Layer : 다른 오브젝트와 겹칠 경우 스프라이트의 순서를 나타낸다. 숫자가 큰 스프라이트가 앞쪽에 위치한다.

챕터 요약 정리

키워드로 정리하기

- Unity의 컴포넌트는 **생명 주기**에 따라 관리되고, 이 과정에서 여러 함수가 호출된다.
- **게임 루프**라고 불리는 무한 반복문에 의해 Unity의 여러 연산이 수행되는데, 반복되는 한 덩어리를 **프레임**이라고 부른다.
- Unity의 오브젝트는 GameObject이고 이름, 태그, 레이어 등의 값을 가진다.
- 씬 내의 오브젝트는 이름, 태그, 컴포넌트 등 여러 방법으로 찾을 수 있다.
- 오브젝트를 에셋으로 만든 것을 **프리팹**이라고 부르며, 일반적인 오브젝트와 동일하게 GameObject이다.
- Instantiate를 이용해 프리팹을 원본으로 오브젝트를 동적으로 생성할 수 있다.
- 씬이 전환되면 기존 씬의 모든 데이터는 초기화된다. 따라서 이전의 씬을 다시 로드하더라도 기존의 데이터는 남아있지 않는다.
- DontDestroyOnLoad가 설정된 오브젝트는 씬이 전환되어도 삭제되지 않고 유지된다.

이미지로 정리하기



위 이미지는 몇 가지 주요 생명 주기 함수의 흐름을 나타낸 것이다.

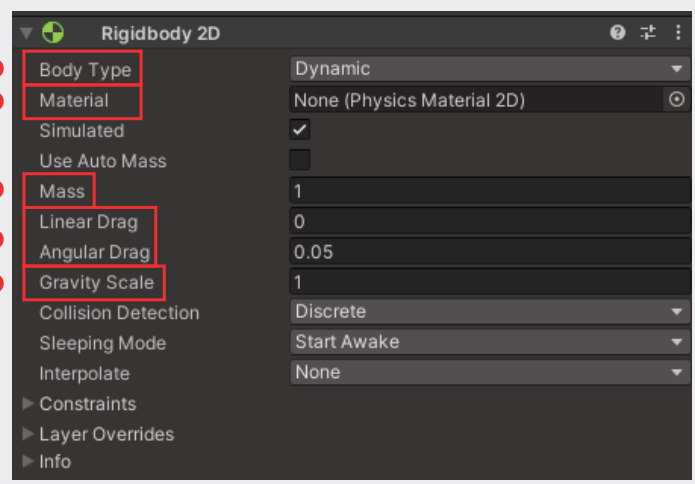
① Awake: 오브젝트가 생성된 직후 Start보다 먼저 호출된다. 컴포넌트가 비활성화 상태여도 호출되며, 주로 GetComponent처럼 초깃값을 할당하는 용도로 쓰인다.
② OnEnable: 활성화 상태가 될 때마다 호출된다.
③ Start: 모든 Awake가 끝난 후 호출된다. 비활성화 상태인 컴포넌트는 호출되지 않는다.
④ FixedUpdate: 일정한 간격으로 실행됨이 보장되는 특수한 업데이트다. 주로 물리 효과에 사용된다.
⑤ Update: 게임의 매 프레임마다 호출된다. 실행 환경에 따라 호출 횟수가 달라지며 입력 처리를 문제없이 수행할 수 있다.
⑥ LateUpdate: 모든 Update가 호출된 이후에 호출된다.
⑦ OnApplicationQuit: 프로그램 종료 전 모든 곳에서 호출된다.
⑧ OnDisable: 비활성화 상태가 될 때마다 호출된다.
⑨ OnDestroy: 한 프레임의 모든 Update가 끝난 후 삭제될 오브젝트 또는 컴포넌트의 OnDestory가 호출된다.

챕터 요약 정리

키워드로 정리하기

- **FixedUpdate**는 일정한 간격마다 호출됨이 보장되는 메소드로 물리 엔진의 기반이다.
- **리지드바디** 컴포넌트가 있는 오브젝트는 물리 효과를 받고, 물리적인 크기와 형태는 **콜라이더**에 의해 정의된다.
- 물리적인 충돌은 **Collision**, 물리적인 충돌 없이 두 오브젝트가 닿는 것은 **Trigger**이다.
- 콜라이더는 **Circle**, **Capsule**, **Polygon** 등 여러 종류가 있고, 각각 특징과 연산량이 다르다.
- 추가적인 물리 효과는 여러 컴포넌트에 나눠져 있다. 대표적으로 **조인트**와 **이펙터**가 있다.

이미지로 정리하기



리지드바디는 오브젝트에 물리 효과를 부여하는 컴포넌트이다.

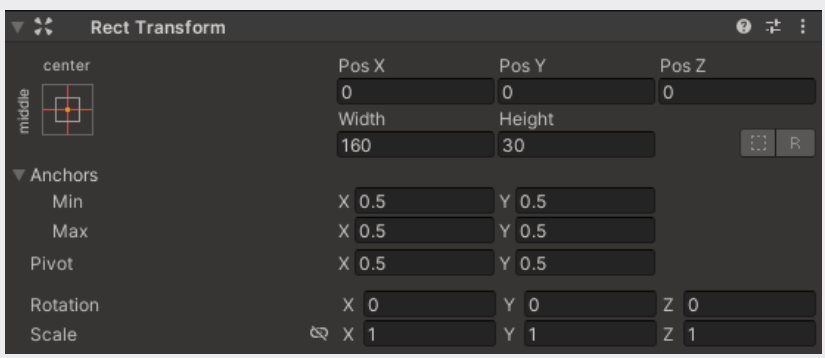
① Body Type: 물체의 종류를 나타낸다. 크게 움직일 수 있는 Dynamic과 특정 위치에 고정된 Static으로 구분할 수 있다.
② Material: 재질의 물리적 특성을 설정한다.
③ Mass: 물체가 갖는 질량이다.
④ Drag: 물체가 갖는 저항이다.
⑤ Gravity Scale: 물체가 중력에 영향을 받는 계수이다.

챕터 요약 정리

키워드로 정리하기

- **UI 오브젝트**는 캔버스 위에 그려진다. 캔버스는 화면, 카메라 또는 월드 좌표계에 그려질 수 있다.
- UI 오브젝트는 **RectTransform**을 가진다. 이는 앵커를 통해 다양한 크기와 위치에 UI를 배치할 수 있게 한다.
- **앵커**는 점, 선, 면의 형태를 가질 수 있고, 그에 맞춰 RectTransform의 크기나 위치가 변화한다.
- UI 오브젝트는 고유하거나 공통된 **이벤트**를 발생시킨다. 발생한 이벤트는 특정 메소드를 호출하거나 변수를 설정한다.

이미지로 정리하기



RectTransform은 위치와 회전, 크기에 대한 정보를 갖던 트랜스폼에 더해 여러 해상도에서도 적절한 위치에 UI를 배치할 수 있도록 앵커와 피봇, 그리고 상대적인 위치와 크기가 추가된 컴포넌트다.

① Pos, Width, Height, ...: UI 오브젝트의 크기와 위치를 지정하는 값이다.
② 피봇: Ui 오브젝트의 중심이다.
③ 앵커: UI 오브젝트가 배치될 기준을 나타낸다. 점, 선, 면의 형태를 가진다.

챕터 요약 정리

01 컴퓨터와 프로그래밍 언어, 그리고 C#

키워드로 정리하기

- **프로그램**은 컴퓨터가 실행해야 하는 연산의 순서와 방법을 나타낸 것이며, 이는 **프로그래밍 언어**로 작성된다.
- C#은 **고급 프로그래밍 언어**이고, Unity의 게임 개발에 사용되는 언어이기도 하다.

02 변수와 상수

키워드로 정리하기

- 변할 수 있는 값을 **변수**, 변할 수 없는 값을 **상수**라고 한다.
- 변수를 사용하기에 앞서 변수의 **선언**과 **초기화**가 필요하다.
- 변수와 상수는 모두 값의 종류를 의미하는 **자료형**이 있다. 자료형은 정수, 실수, 문자열 등으로 나뉜다.
- const가 붙은 변수는 상수로 취급되는 변수로, 수정할 수 없는 변수이다.

예제로 정리하기

```
int i = 0, j; // ①
j = 1; // ②
float f = 1.0f, g = 2.0f; // ③
string s; // ④
s = "Hello"; // ⑤
```

① 정수형 변수 i와 j를 선언하고, i를 0을 초기화한다.
② 선언된 변수 j를 1로 초기화한다.
③ 실수형 변수 f와 g를 선언하고, f를 1, g를 2로 초기화한다.
④ 문자열형 변수 s를 선언한다.
⑤ 선언된 문자열형 변수 s를 "Hello"로 초기화한다.

03 연산자

키워드로 정리하기

- **연산자**는 피연산자에 대해 특정한 연산을 수행하여 결과를 계산한다.
- **산술 연산자**는 더하기, 빼기, 곱하기, 나누기, 나머지 연산을 수행한다.
- **비교 및 논리 연산자**는 주어진 식이 참인지 거짓인지를 계산한다.
- **대입 연산자**는 변수에 값을 저장한다. **복합 대입 연산자**와 **증감 연산자**는 대입 연산자의 특정한 형태를 축약한 것이다.

04 조건문

키워드로 정리하기

- **조건문**은 특정 조건을 만족하거나 만족하지 않을 경우 코드를 실행한다.
- **if문**은 대표적인 조건문으로 **else if**, **else**와 함께 구성될 수 있다.
- **switch문**은 주어진 값이 여러 경우에 대응되는 경우를 나누어 처리할 때 사용된다.
- switch문은 한 번 진입한 이후 **break**를 만나기 전까지의 코드를 모두 실행한다. 모든 조건을 만족하지 않았다면 **default**에서 진입한다.

예제로 정리하기

손으로 익히는 코딩

```
int x = 10;
if (x == 0)
    Console.WriteLine("x == 0"); // ①
else
{
    Console.WriteLine("x != 0"); // ②
    if (x > 0)
        Console.WriteLine("x > 0"); // ③
    else
        Console.WriteLine("x < 0"); // ④
}
```

① x == 0 조건을 만족하는 경우 실행된다.
② x != 0 조건을 만족하는 경우 실행된다.
③ x != 0 && x 〉 0 조건을 만족하는 경우 실행된다.
④ x != 0 && x 〈= 0 조건을 만족하는 경우 실행된다.

05 반복문

키워드로 정리하기

- **반복문**은 특정 조건을 만족하는 코드를 반복적으로 실행한다.
- **while문**은 기본적인 반복문으로, if문과 비슷하지만 조건을 만족하지 않을 때까지 반복한다.
- **for문**은 초기화식, 조건식, 증감식을 한 번에 쓰는 반복문이다.
- **continue**는 현재의 반복을 중단하고 다시 조건을 검사하며, **break**는 반복문 자체를 중단하고 빠져나온다.

더 멋진 내일 Tomorrow 을 위한 내일 My Career

내일은 유니티 Unity

응용 실전편

이우령 지음

김앤북
KIM&BOOK

챕터 요약 정리

키워드로 정리하기

- **삼각함수**는 직각삼각형을 이루는 세 변의 길이와 각도 사이의 관계를 나타내는 함수이다. sin, cos, tan, csc, sec, cot 등이 있다.
- **역삼각함수**는 삼각함수와 반대로 길이 정보를 바탕으로 각도 정보를 구하는 함수이다.
- **벡터**는 크기와 방향을 갖는 값이다. 2차원, 3차원, 4차원 벡터가 주로 사용된다.
- 벡터의 **내적**과 **외적**은 두 벡터 사이의 연산으로, 벡터 사이의 정보를 간단한 연산으로 구할 수 있다.
- Unity는 내부적으로 **쿼터니언**을 이용하여 회전을 계산한다. 여러 함수를 통해 쿼터니언을 계산할 수 있다.
- **오일러 각**은 x, y, z축에 대한 회전으로 전체 회전을 나타내는 방법이다. 쿼터니언과 상호 변환이 가능하다.

챕터 요약 정리

키워드로 정리하기

- 3D 오브젝트는 **메쉬**로 형태를, **머티리얼**로 표면을, **쉐이더**로 형태와 표면, 그리고 빛의 연산을 지정한다.
- 메쉬, 머티리얼, 쉐이더는 모두 에셋이며, 필요에 따라 다양한 메쉬, 머티리얼, 쉐이더를 추가하여 사용한다.
- 머티리얼은 쉐이더에 종속적이다. 쉐이더에 필요한 속성을 머티리얼에서 정의한다.
- 3D 오브젝트에서 주로 적용되는 컴포넌트로는 **3D 물리 컴포넌트**, **터레인** 등이 있다.
- 3D 렌더링은 연산의 비용이 높으므로 다양한 최적화 기법이 사용된다. **컬링**, **LOD**, **배칭** 등이 있다.

챕터 요약 정리

키워드로 정리하기

- **코루틴**은 실행 중간에 일시 정지가 가능하며, 정지를 풀고 멈췄던 부분에서 다시 시작할 수 있는 함수를 말한다. 메인 스레드에서 실행되며, 컴포넌트의 생명주기에 맞춰 동작한다.
- **비동기**는 작업이 완료될 때까지 기다리지 않고, 다른 작업을 동시에 수행하는 것을 말한다. 이러한 방식으로 프로그램을 만드는 것을 비동기 프로그래밍이라고 한다.
- **스레드**는 프로그램 내에서 동시에 실행될 수 있는 가장 작은 실행 단위이다.
- 일반적으로 여러 스레드는 하나의 값에 동시에 접근하게 두어서는 안 된다. 스레드 사이의 데이터 전달을 위해서는 Concurrent **컬렉션**을 사용한다.

예제로 정리하기

손으로 익히는 코딩

```
using System.Threading;

public class MyComp : MonoBehaviour
{
    private Thread myThread; // ①
    void Func() // ②
    {
        for (int i = 0; i < 10; i++)
        {
            Debug.Log("Thread");
            Thread.Sleep(1000); // ③
        }
    }
    void Start()
    {
        myThread = new Thread(new ThreadStart(Func));
        myThread.Start(); // ④
    }
}
```

① 스레드는 Thread 객체로 관리된다.

② 스레드는 특정 함수를 실행하도록 만들어진다.

③ Thread.Sleep은 현재 스레드를 잠시 멈추게 한다.

④ 스레드는 실행 함수로 새로운 Thread 객체를 만든 후 시작한다.

챕터 요약 정리

키워드로 정리하기

- 메인 메모리는 프로그램이나 컴퓨터가 종료되면 데이터가 초기화되므로, HDD나 SSD와 같은 보조 기억 장치에 데이터를 저장해야 한다.
- **PlayerPrefs**는 Key-Value로 간단한 데이터를 저장할 수 있도록 만들어진 Unity의 데이터 입출력 기능이다.
- 파일을 읽고 쓰기 위해서는 **StreamReader**, **StreamWriter**, **FileStream** 등을 이용한다.
- **텍스트 파일**은 텍스트로 이루어진 파일이다. txt, cs, json, xml 등의 파일이 있다.
- 데이터 저장은 **인터페이스**로 만들거나 **계층 구조**로 구성하는 등 여러 기법을 추가할 수 있다.
- 서로 다른 버전의 파일을 읽기 위해서는 **파일의 버전**을 기록하거나 파일의 형식을 맞춰야 한다.

예제로 정리하기

```
using System.IO;

void SaveJSON()
{
    MyData data = new MyData(); // ①
    data.intData = 7;
    data.floatData = 4.4f;
    data.stringDatas = new string[] { "Hello", "World"};
    string jsonData = JsonUtility.ToJson(data); // ②

    StreamWriter sw = new StreamWriter("data.json"); // ③
    sw.Write(jsonData);
    sw.Close(); // ④
}
```

① Serializable 객체는 JsonUtility를 이용해 JSON 형태로 직렬화할 수 있다.
② JsonUtility는 객체를 JSON으로 직렬화하거나 반대로 역직렬화하는 기능을 제공한다.
③ 파일 입출력은 StreamWriter, FileStream 등의 객체를 이용한다.
④ 파일에 대한 처리가 끝나면 파일을 닫아야 한다.

챕터 요약 정리

01 객체지향

키워드로 정리하기

- 프로그래밍 언어는 저마다의 문제를 해결하는 방법인 **패러다임**을 가지고 설계 및 구현되었다.
- **객체지향 프로그래밍**은 프로그램을 구성하는 객체로 문제를 해결하는 패러다임이다.
- **객체**는 사물이나 동물처럼 눈에 보이는 것일 수도 있고, 추상적인 개념일 수도 있다.
- **클래스**는 객체들의 공통된 속성과 행동, 상호작용을 모은 것이다. 일종의 분류 역할을 한다.

02 클래스와 인스턴스

키워드로 정리하기

- 클래스는 class로 만들고, new 키워드로 클래스로부터 인스턴스를 생성할 수 있다.
- 클래스에는 속성을 나타내는 **멤버 변수**와 행동 및 상호작용을 나타내는 **메소드**가 있다.
- **생성자**는 클래스가 생성되는 시점에 자동으로 호출되는 메소드이다.
- 메소드는 함수를 실행하는 주체가 있다. 이때 주체는 this로 나타내며, 이름이 겹쳐지지 않는다면 this를 생략하고 멤버 변수나 메소드에 접근할 수 있다.

예제로 정리하기

손으로 익히는 코딩

```
public class Animal // ①
{
    public string name; // ②
    public Animal(string name) // ③
    {
        this.name = name; // ④
    }
    public Animal() // ⑤
    {
        this.name = "animal";
    }
}
```

① 클래스는 class로 만들고, 중괄호 안에 멤버 변수와 메소드를 추가한다.
② 멤버 변수는 객체가 가지는 변수이자 속성이다.
③ 메소드는 객체가 주체가 되는 함수이다. 생성자는 객체가 만들어질 때 호출되는 메소드이다.
④ 메소드 내에서 자기 자신에 접근하기 위해서는 this를 사용한다.
⑤ 메소드는 매개 변수에 따라 구분되는데, 이를 메소드 오버로딩이라고 한다.

03 상속과 엑세스 한정자

키워드로 정리하기

- **상속**은 다른 클래스의 멤버 변수와 메소드를 물려받아 새로운 클래스를 만드는 방법이다.
- 다른 클래스를 상속받으면 그 클래스가 가지고 있는 멤버 변수와 메소드를 그대로 사용할 수 있고, 그 클래스를 저장하는 변수에도 객체를 담을 수 있다.
- 클래스는 외부 및 자식 클래스에서의 접근을 제한하는 **엑세스 한정자**가 있다. 대표적으로 private, protected, public이 있다.
- 상속받은 클래스는 기본적으로 부모 클래스의 기본 생성자를 호출한다. 이를 바꾸기 위해서는 생성자 옆에 base로 부모 클래스의 생성자를 지정한다.
- **메소드 오버라이딩**은 부모 클래스의 **가상 메소드**를 자식 클래스에서 덮어씌우는 기능이다.

예제로 정리하기

손으로 익히는 코딩

```
public class Shape
{
    protected string name; // ①
    public Shape(string name)
    {
        this.name = name;
    }
}
public class Circle : Shape // ②
{
    private float radius;
    public Circle(float radius, string name) :
        base(name) // ③
    {
        this.radius = radius;
```

```
    }
    public PrintInfo()
    {
        Console.WriteLine(name); // ④
        Console.WriteLine(radius);
    }
}
```

① 엑세스 한정자는 멤버 변수와 메소드의 접근을 제한한다. protected는 자신과 자식 클래스에서의 접근만을 허가한다.
② 상속은 부모 클래스로부터 멤버 변수와 클래스를 물려받는다.
③ 자식 클래스의 생성자에서 부모 클래스의 생성자를 지정하여 호출할 수 있다.
④ 자식 클래스는 물려받은 부모 클래스의 멤버 변수를 사용할 수 있다. 단, 엑세스 한정자가 private이면 불가하다.

04 배열과 컬렉션

키워드로 정리하기

- **배열**은 동일한 종류의 값을 저장하는 객체다. 배열을 생성하는 시점에서 배열의 크기가 결정된다.
- 배열은 원소에 순서가 있고, 이 순서를 **인덱스**라고 한다. 배열의 원소는 대괄호로 인덱스를 써서 접근한다.
- **컬렉션**은 많은 원소를 저장하되, 특별한 기능이나 저장 방식이 있거나 제약 조건이 추가된 객체들이다.
- foreach는 배열과 컬렉션에서 사용할 수 있는 반복문으로, 객체가 가지는 모든 원소를 순회한다.

예제로 정리하기

손으로 익히는 코딩

```
int[] ints = new int[100]; // ①
for (int i = 0; i < 100; i++) // ②
    ints[i] = i; // ③
foreach (int x in ints) // ④
    Console.WriteLine(x);
```

① 배열은 같은 종류의 데이터를 정해진 크기만큼 담을 수 있는 객체이다.
② 배열은 반복문과 함께 쓰이는 경우가 많다.

③ 배열은 인덱스라 불리는 원소의 순서로 원소에 접근한다.
④ foreach는 배열이나 컬렉션처럼 여러 데이터를 가지는 객체를 순회하는 반복문이다.

05 정적 변수와 메소드

키워드로 정리하기

- **정적 변수**와 **정적 메소드**는 객체가 아닌 클래스가 가지는 변수와 메소드이다.
- 정적 변수는 프로그램 내에서 유일하게 하나의 값만을 가진다.
- 정적 메소드는 메소드를 호출하는 주체가 없으므로 멤버 변수를 사용할 수 없다.
- **정적 객체**는 정적 메소드와 정적 변수만을 포함하며, 객체를 생성할 수 없다.

예제로 정리하기

```
public static class Math // ①
{
    public static float PI = 3.141592f; // ②
    public static float GetCircleArea(float radius) // ③
    {
        return PI * radius * radius;
    }
}
```

① 정적 클래스는 class 앞에 static이 붙고, 객체를 생성할 수 없다.
② 정적 변수는 모든 객체가 공통으로 사용하는 변수에 쓰인다.
③ 정적 메소드는 메소드의 주체가 필요하지 않은 경우에 쓰인다.

06 추상 클래스와 인터페이스

키워드로 정리하기

- **추상 클래스**는 객체를 만들 수 없는 클래스로, **추상 메소드**를 포함한다.
- 추상 클래스를 상속받은 클래스는 모든 추상 클래스를 구현해야 한다.
- **인터페이스**는 능력, 용도, 상호작용을 나타내는 기능이다.
- 클래스는 최대 하나만 상속받을 수 있지만 인터페이스 구현에는 제한이 없다.

예제로 정리하기

```
public abstract class Animal // ①
{
    public abstract void Move(); // ②
}
public class Dog : Animal // ③
{
    public override void Move() // ④
    {
        Console.WriteLine("Dog.Move");
    }
}
```

① 추상 클래스는 객체를 만들 수 없는 클래스이다.
② 추상 클래스는 추상 메소드를 포함할 수 있다. 추상 클래스에서는 추상 메소드를 구현하지 않는다.
③ 다른 클래스는 추상 클래스를 상속받아 사용한다.
④ 추상 클래스를 상속받았다면 모든 추상 메소드를 구현해야 한다.

예제로 정리하기

손으로 익히는 코딩

```
int i = 10;
while (i >= 0) // ①
{
    if (i % 2 == 0)
    {
        i--;
        continue; // ②
    }
    if (i == 3)
        break; // ③
    Console.WriteLine(i);
    i--;
}
```

① while문은 주어진 조건을 만족하는 한 계속 반복한다. 조건이 거짓이면 반복을 종료한다.

② continue는 현재의 반복을 종료하고 다시 반복문의 조건을 검사한다.

③ break는 반복문을 완전히 종료한다.

06 함수

키워드로 정리하기

- **함수**는 하나의 프로그램을 더 작은 기능 단위로 나눈 것으로 서브 프로그램이라고도 한다. 함수는 0개 이상의 데이터를 입력으로 받아 0개 또는 1개의 출력을 반환한다.
- 함수는 **반환형, 함수명, 매개 변수, 정의부**로 구성된다.
- 자기 자신을 호출하는 함수를 **재귀 함수**라고 한다.
- 재귀 함수는 자기 자신을 호출하는 **재귀 호출**, 그리고 재귀 호출이 끝나는 **종료 조건**이 필요하다.

예제로 정리하기

손으로 익히는 코딩

```
int Fibo(int n) // ①
{
    if (n <= 2) // ②
        return 1; // ③
    return Fibo(n - 1) + Fibo(n - 2); // ④
}
Console.WriteLine(Fibo(5)); // ⑤
```

① 함수는 반환형, 함수명, 매개 변수, 정의부로 구성된다.
② 재귀 함수는 종료 조건이 필요하다.
③ return은 함수를 끝내고 계산 결과를 호출한 곳에 반환한다.
④ 재귀 함수는 자기 자신을 호출하는 재귀 호출이 있다.
⑤ 함수의 호출은 함수의 이름 옆에 소괄호로 함수의 입력에 해당하는 값을 넣는다.