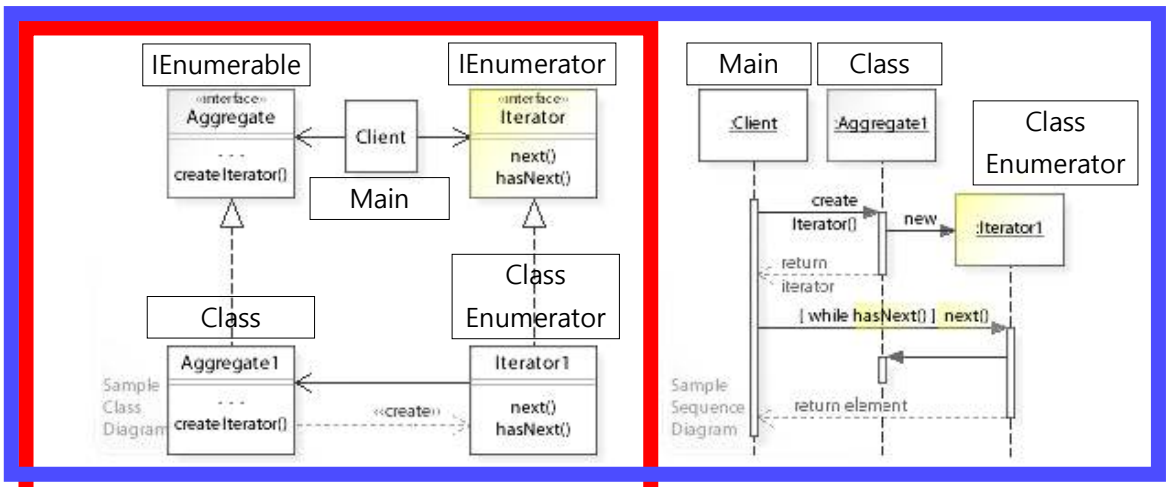
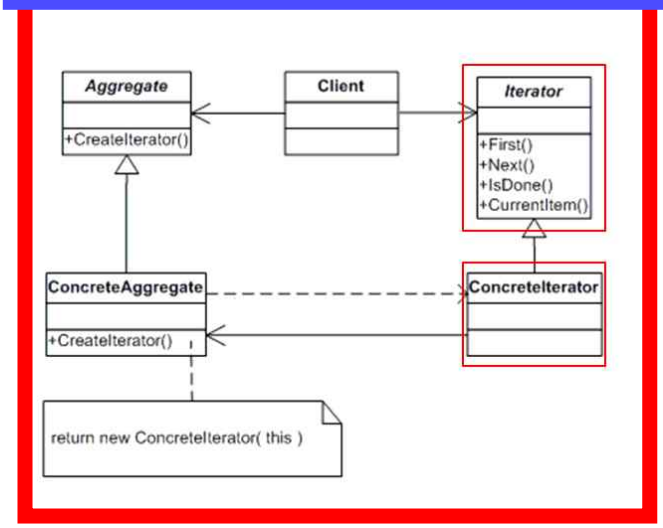
기술면접 – 반복자 패턴(Iterator Pattern)

반복자 패턴

- **컨테이너를 가로지르는 객체를 통해 컨테이너의 요소들을 순회할 수 있도록 해준다.**
- 반복자 (Iterator) 패턴은 **컨테이너에 대해 순회할 수 있는 객체를 정의**하고 이를 통해 **컨테이너의 모든 요소를 순회**할 수 있게 하는 패턴이다. 클라이언트는 요소의 순회에 대한 알고리즘을 직접 구현하는 대신에 해당 컨테이너가 제공하는 반복자 객체의 인터페이스를 통해 **어떤 컨테이너에 대해서라도 동일한 방법으로 간편하게 요소를 순회**할 수가 있다.
- 반복자 패턴을 사용하면 **요소의 순회에 대한 알고리즘을 컨테이너로부터 분리시켜 확장성이 용이**해지고, **컨테이너의 직접적인 노출 없이** 반복자 객체를 통해 요소에 대한 순회를 수행할 수 있게 해 준다. 또한 하나의 컨테이너에 여러 반복자 객체를 구현함으로써 **컨테이너에 대한 다양한 순회 방법을 제공**할 수 있다.
- **C# 에서는 컨테이너에 대한 순회 방법을 추상화한 IEnumerator, IEnumerable 인터페이스를 제공하고 있다.**





위 그림 구현

```
public class PatternIterator
{
    public interface IAggregate
    {
        IIterator CreateIterator();
    }

    public class Aggregate : IAggregate
    {
        private string[] _strings;

        public string[] strings
        {
            get
            { return _strings; }
        }

        public Aggregate(string[] strings)
        {
            _strings = strings;
        }

        public IIterator CreateIterator()
        {
            return new Iterator(this);
        }
    }

    public interface IIterator
    {
        string Next();
        bool HasNext();
    }

    public class Iterator : IIterator
    {
        private Aggregate _aggregate;

        private int _index;

        public Iterator(Aggregate aggregate)
        {
            _aggregate = aggregate;

            _index = 0;
        }

        public string Next()
        {
```

```
            return _aggregate.strings[_index++];
        }

        public bool HasNext()
        {
            return _index < _aggregate.strings.Length;
        }
    }

    public static void Main(string[] args)
    {
        string[] strings = new string[]
        {
            "string A",
            "string B",
            "string C",
            "string D",
            "string E"
        };

        IAggregate aggregate = new Aggregate(strings);

        IIterator iterator = aggregate.CreateIterator();

        while (iterator.HasNext())
        {
            Console.WriteLine(iterator.Next());
        }
    }
}
```

위의 예제에서 클라이언트는 생성한 Aggregate 객체에서 제공받은 Iterator 객체를 통해 Aggregate 객체의 모든 요소를 순회하고 있다. while 문에서 Iterate 객체의 HasNext() 메소드가 false를 반환할 때 까지 Next() 메소드를 통해 Aggregate 객체의 다음 요소를 반복해서 가져오고 있다.

코루틴

- 코루틴이란 **어떠한 시간이나 조건식이 만족하기 전까지는 멈춰있다가 만족할 경우 기억하고 있던 이전의 작업들을 이어서 하는 함수**입니다.
- 코루틴은 실행을 중지하여 Unity에 제어권을 돌려주고, 그러나 계속할 때 는 다음 플에임에서 중지한 곳부터 실행을 계속할 수 있는 기능입니다. 함수를 호출하면 반환 값을 돌려주기 반환하기 전에 실행 완료됩니다.
- 단일쓰레드로 동작하는 유니티를 **멀티쓰레드처럼 비슷하게** 작동하게 해주는 것이 코루틴입니다.
- C#은 **IEnumerator를 반환하는 코루틴을 이터레이터 인터페이스로서 프로그래머가 IEnumerator를 구현할 수 있게 도와주고**, 진입하는 지점이 한 개인 보통의 서브루틴 방식과는 다르게 진입하는 지점까지 여러 개를 가질 수 있습니다.
- **yield return 을 통해 바로 전 시점을 기억하고 다음 호출하게 될 때 그 다음부터 실행이 되도록 합니다.**

코루틴을 알기 위해서 기본적으로 알아야 하는 함수.
서브루틴 : 일반적인 함수에서 사용되는 작동개념으로서 함수는 실행될 때 진입하는 지점을 저장하고, return 구문에 의해서 종료되는 지점을 저장합니다.
IEnumerable 콜랙션 : foreach구문 등에서 개체를 한 개씩 넘겨주는 일을 합니다.
IEnumerator : 지금까지 몇번째를 읽었는지(State)를 기억해서 MoveNext()를 호출받으면 다음순번으로 이동, Current를 요구할 때 해당 순번의 개체를 리턴합니다.
반복기(iterator) : 원소들을 순회 할 수 있는 객체 입니다.