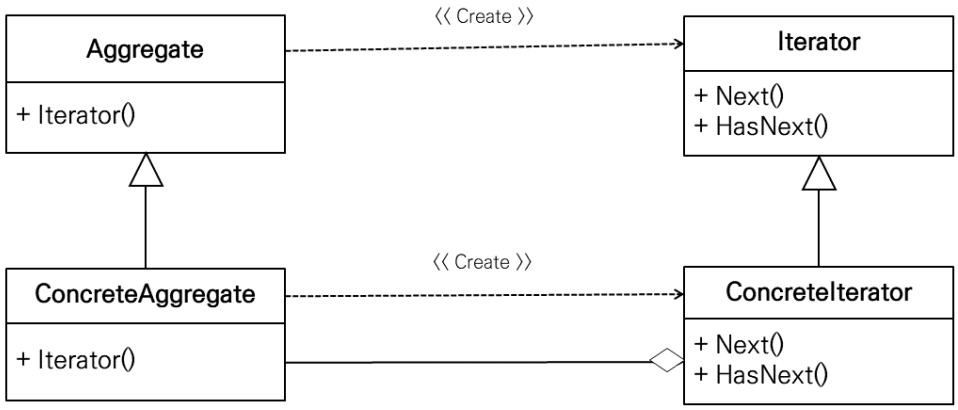
1. 이터레이터 / 반복자 패턴(Iterator Pattern)

1) 이터레이터 / 반복자 패턴

- 행동 패턴
- 반복이 필요한 자료구조들을 모두 동일한 인터페이스를 통해 접근할 수 있도록 메서드를 이용해 자료 구조를 활용할 수 있도록 함
- 구현 방법을 노출시키지 않으면서 그 집합체 안에 들어있는 모든 항목에 접근할 수 있게 해주는 방법을 제공하는 디자인 패턴
- 각 항목에 일일이 접근할 수 있게 해주는 기능을 집합체가 아닌 반복자 객체에서 책임지게 되어 집합체 인터페이스 및 구현이 간단해짐
- std::vector::iterator도 반복자 패턴

2) 클래스 다이어그램

Iterator Pattern_Class



3) 예제

```
#pragma once

template<typename T>
class Node
{
public:
	T m_data;
	Node<T>* m_next;

	Node(const T& set, Node<T>* next)
	{
		m_data = set;
		m_next = next;
	}
};


template<typename T2>
class CIterator
{
private:
	Node<T2>* m_cur;

public:
	CIterator(Node<T2>*ptr = nullptr) : m_cur(ptr) { }

	CIterator& operator++()
	{
		m_cur = m_cur->m_next;
		return *this;
	}

	T2& operator*()
	{
		return m_cur->m_data;
	}

	bool operator==(const CIterator &ref)
	{
		return m_cur == ref.m_cur;
	}

	bool operator!=(const CIterator &ref)
	{
		return m_cur != ref.m_cur;
	}
};
```

```
template<typename T3>
class CList
{
private:
	Node<T3>* m_head;

public:
	CList() : m_head(nullptr) { }
	~CList() { if (m_head) delete m_head; }

	void PushFront(const T3& a)
	{
		m_head = new Node<T3> (a, m_head);
	}
	typedef CIterator<T3> m_iterator;

	m_iterator begin()
	{
		return m_iterator(m_head);
	}

	m_iterator end()
	{
		return m_iterator(nullptr);
	}
};



// ============================================================================


#include <iostream>
#include "Iterator.h"

using namespace std;

int main()
{
	CList<int> list;
	list.PushFront(10);
	list.PushFront(20);
	list.PushFront(30);

	CList<int>::m_iterator it;

	for (it = list.begin(); it != list.end(); ++it)
	{
		cout << *it << endl;
	}
}
```