# 17. 오브젝트 리스트와 프로토타입

Ref

```
class CRef
{
protected:
        CRef();
        virtual ~CRef() = 0;

protected:
        int             m_iRef;
        bool    m_bEnable;
        bool    m_bLife;
```

Layer

```
class CLayer
{
private:
        friend class CScene;

private:
        CLayer();

public:
        ~CLayer();

private:
        class CScene*           m_pScene;
        string                          m_strTag;
        int                                     m_iZOrder;
        list<class CObject*>    m_ObjList;
        bool                    m_bEnable;
        bool                    m_bLife;
```

Ref와 Layer에 껏다 키는 기능을 하는 m_bEnable, 소멸 여부를 나타내는 m_bLife 변수를 만들어준다.

## Scene

```
CScene::~CScene()
{
    Safe_Delete_VecList( m_LayerList );
}
```

```
void CScene::Input( float fDeltaTime )
{
    list<CLayer*>::iterator iter;
    list<CLayer*>::iterator iterEnd = m_LayerList.end();

    for( iter = m_LayerList.begin(); iter != iterEnd; )
    {
        if( !(*iter)->GetEnable() )
        {
            ++iter;
            continue;
        }

        (*iter)->Input( fDeltaTime );

        if( !(*iter)->GetLife() )
        {
            SAFE_DELETE( (*iter) );
            iter = m_LayerList.erase( iter );
            iterEnd = m_LayerList.end();
        }

        else
            ++iter;
    }
}

int CScene::Update( float fDeltaTime )
{
    list<CLayer*>::iterator iter;
    list<CLayer*>::iterator iterEnd = m_LayerList.end();

    for( iter = m_LayerList.begin(); iter != iterEnd; )
    {
        if( !(*iter)->GetEnable() )
        {
            ++iter;
            continue;
        }

        (*iter)->Update( fDeltaTime );

        if( !(*iter)->GetLife() )
        {
            SAFE_DELETE( (*iter) );
            iter = m_LayerList.erase( iter );
            iterEnd = m_LayerList.end();
        }

        else
            ++iter;
    }
    return 0;
```

## Layer

```
CLayer::~CLayer()
{
        list<CObject*>::iterator iter;
        list<CObject*>::iterator iterEnd = m_ObjList.end();

        for( iter = m_ObjList.begin(); iter != iterEnd; ++iter)
        {
                CObject::EraseObj( *iter );
                SAFE_RELEASE( (*iter) );
        }

        m_ObjList.clear();
}

void CLayer::AddObject( CObject* pObject )
{
        pObject->SetScene( m_pScene );
        pObject->SetLayer( this );
        pObject->AddRef();

        m_ObjList.push_back( pObject );
}

void CLayer::Input( float fDeltaTime )
{
        list<CObject*>::iterator iter;
        list<CObject*>::iterator iterEnd = m_ObjList.end();

        for( iter = m_ObjList.begin(); iter != iterEnd; )
        {
                if( !(*iter)->GetEnable() )
                {
                        ++iter;
                        continue;
                }

                (*iter)->Input( fDeltaTime );

                if( !(*iter)->GetLife() )
                {
                        CObject::EraseObj( *iter );
                        SAFE_RELEASE( (*iter) );
                        iter = m_ObjList.erase( iter );
                        iterEnd = m_ObjList.end();
                }

                else
                        ++iter;
        }
}

int CLayer::Update( float fDeltaTime )
```

Scence은 Layer에 대해, Layer는 Object들에 대해 m_bLife 변수 상태를 체크해 소멸상태이면 SAFE_DELETE 또는 SAFE_RELEASE로 제거, 각 리스트에서도 제거 해준다.

# Object

```h
#pragma once

#include "../Game.h"
#include "../Ref.h"
#include "../Scene/Layer.h"

class CObject : public CRef
{
protected:
	CObject();
	CObject( const CObject& obj );
	virtual ~CObject();

private:
	static list<CObject*> m_ObjList;
	static unordered_map<string, CObject*> m_mapPrototype;

public:
	static void AddObj( CObject* pObj );
	static CObject* FindObject( const string& strTag );
	static void EraseObj( CObject* pObj );
	static void EraseObj( const string& strTag );
	static void EraseObj();
	static void ErasePrototype( const string& strTag );
	static void ErasePrototype();

private:
	static CObject* FindPrototype( const string& strKey );
```

```h
	virtual void Input( float fDeltaTime );
	virtual int Update( float fDeltaTime );
	virtual int LateUpdate( float fDeltaTime );
	virtual void Collision( float fDeltaTime );
	virtual void Render( HDC hDC, float fDeltaTime );
	virtual CObject* Clone() = 0;

public:
	template <typename T>
	static T* CreateObj( const string& strTag, class CLayer* pLayer = nullptr )
	{
		T* pObj = new T;

		pObj->SetTag( strTag );

		if( !pObj->Init() )
		{
			SAFE_RELEASE( pObj );
			return nullptr;
		}

		if( pLayer )
		{
			pLayer->AddObject( pObj );
		}

		AddObj( pObj );

		return pObj;
	}

	template <typename T>
	static T* CreatePrototype( const string& strTag )
	{
		T* pObj = new T;

		pObj->SetTag( strTag );

		if( !pObj->Init() )
		{
			SAFE_RELEASE( pObj );
			return nullptr;
		}

		pObj->AddRef();
		m_mapPrototype.insert( mae_pari( strTag, pObj ) );

		return pObj;
	}

	static CObject* CreateCloneObj( const string& strPrototypeKey, const string& strTag, CLayer* pLay
};
```

복사를 위한 함수 순수 가상함수로 선언

```cpp
void CObject::AddObj( CObject* pObj )
{
	pObj->AddRef();
	m_ObjList.push_back( pObj );
}

CObject* CObject::FindObject( const string& strTag )
{
	list<CObject*>::iterator iter;
	list<CObject*>::iterator iterEnd = m_ObjList.end();

	for( iter = m_ObjList.begin(); iter != iterEnd; ++iter )
	{
		if( (*iter)->GetTag() == strTag )
		{
			(*iter)->AddRef();
			return *iter;
		}
	}

	return nullptr;
}

CObject* CObject::FindPrototype( const string& strKey )
{
	auto iter = m_mapPrototype.find( strKey );

	if( iter == m_mapPrototype.end() )
	{
		return nullptr;
	}

	return iter->second;
}

void CObject::EraseObj( CObject* pObj )
{
	list<CObject*>::iterator iter;
	list<CObject*>::iterator iterEnd = m_ObjList.end();

	for( iter = m_ObjList.begin(); iter != iterEnd; ++iter )
	{
		if( *iter == pObj )
		{
			SAFE_RELEASE( (*iter) );
			iter = m_ObjList.erase( iter );
			return;
		}
	}
}
```

```cpp
void CObject::EraseObj( const string& strTag )
{
	list<CObject*>::iterator iter;
	list<CObject*>::iterator iterEnd = m_ObjList.end();

	for( iter = m_ObjList.begin(); iter != iterEnd; ++iter )
	{
		if( (*iter)->GetTag() == strTag )
		{
			SAFE_RELEASE( (*iter) );
			iter = m_ObjList.erase( iter );
			return;
		}
	}
}

void CObject::EraseObj()
{
	Safe_Release_VecList( m_ObjList );
}

void CObject::ErasePrototype( const string& strTag )
{
	auto iter = m_mapPrototype.find( strTag );

	if( !iter->second )
		return;

	SAFE_RELEASE( iter->second );
	m_mapPrototype.erase( iter );
}

void CObject::ErasePrototype()
{
	Safe_Release_Map( m_mapPrototype );
}
```

```cpp
CObject* CObject::CreateCloneObj( const string& strPrototypeKey, const string& strTag, CLayer* pL
{
	CObject* pProto = FindPrototype( strPrototypeKey );

	if( !pProto ) return nullptr;

	CObject* pObj = pProto->Clone();

	pObj->SetTag( strTag );

	if( pLayer )
	{
		pLayer->AddObject( pObj );
	}

	AddObj( pObj );

	return pObj;
}
```

오브젝트의 경우 다양한 유형의 오브젝트 원본을 만들어 놓고 이를 복사해서 활용하도록 한다. HDD에 있는 데이터를 사용시 가져와서 쓰는건 효율이 떨어지고, 이를 미리 램 공간에 올려놓고 활용하기 위함이다.

## Object

```
public:
        virtual bool Init() = 0;
        virtual void Input( float fDeltaTime );
        virtual int Update( float fDeltaTime );
        virtual int LateUpdate( float fDeltaTime );
        virtual void Collision( float fDeltaTime );
        virtual void Render( HDC hDC, float fDeltaTime );
        virtual CObject* Clone() = 0;
```

## MoveObject

```
public:
        virtual bool Init() = 0;
        virtual void Input( float fDeltaTime );
        virtual int Update( float fDeltaTime );
        virtual int LateUpdate( float fDeltaTime );
        virtual void Collision( float fDeltaTime );
        virtual void Render( HDC hDC, float fDeltaTime );
        virtual CMoveObject* Clone() = 0;
};
```

## StaticObject

```
public:
        virtual bool Init() = 0;
        virtual void Input( float fDeltaTime );
        virtual int Update( float fDeltaTime );
        virtual int LateUpdate( float fDeltaTime );
        virtual void Collision( float fDeltaTime );
        virtual void Render( HDC hDC, float fDeltaTime );
        virtual CStaticObject* Clone() = 0;
};
```

## Player

```
CPlayer* CPlayer::Clone()
{
    return new CPlayer(*this);
}
```

## Minion

```
CMinion* CMinion::Clone()
{
        return new CMinion( *this );
}
```

순수 가상 함수 구현