

1. 클래스 다이어그램의 구성요소 정리

[클래스]

형식	설명
<div> <div>Class Name</div> <div>- attributeName : type</div> <div>+ operationName() : type</div> </div>	<p>보통 이름, 멤버 변수, 멤버 함수의 3구역으로 나누어서 표기함.</p> <p>멤버 변수와 멤버 함수는 생략 가능, 이름은 생략 불가</p>

[멤버 변수]

형식	
+ AttributeName : TypeName [*]	
표시 여부	+ : Public - : Private # : Protected ~ : Package
형식이 존재 하지 않음	: TypeName 생략
다중성	[*] (단, 1이면 생략)
final 키워드 사용 상수	{ readOnly }
static	<u>해당 변수 밑에 밑줄</u>

[멤버 함수]

형식	
+ OperationName(parameter1 : Type 1[*], ...) : ReturnType [*]	
표시 여부	+ : Public - : Private # : Protected ~ : Package
virtual 선언	1) <i>이탤릭체 표기</i> 2) ReturnType 뒤에 { abstract } 표기
반환 형식	: ReturnType 정의되어 있지 않을 경우 생략
다중성	[*] (단, 1이면 생략)
static	<u>해당 함수 밑에 밑줄</u>

[다중성]

하나의 인스턴스에 연관된 다른 쪽 클래스의 가능한 인스턴스의 수 의미

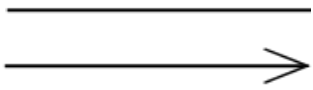
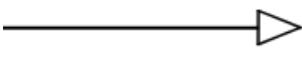
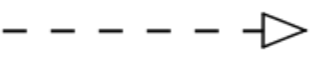
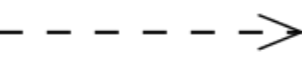
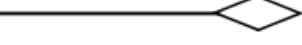
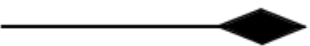
다중성	* == : 무제한 1, 2, 4 : 1, 2, 4 1..* : 1 이상 5..10 : 5 ~ 10 1,2...4 : 1, 2 ~ 4
-----	--

[스테레오 타입]

UML 제공 요소 이외의 추가적인 확장 요소 표기. 길러멧 사이에 작성. 이름 부분에 작성됨.

순수 가상 클래스	« interface »
추상 클래스	« abstract », { abstract }

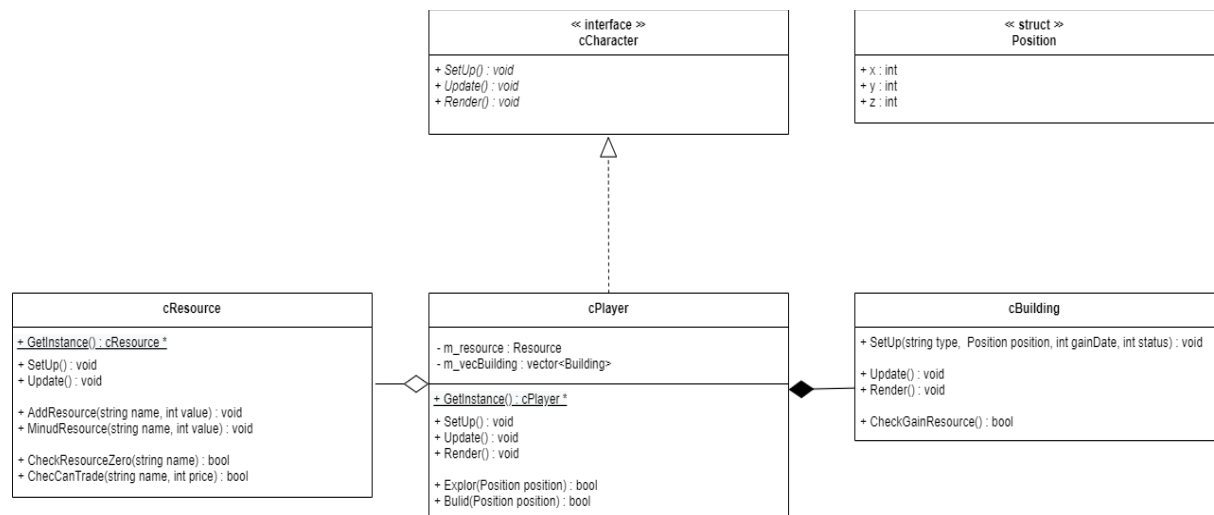
[관계]

형식	이름	설명
	Association (연관)	한 클래스가 다른 클래스에서 제공하는 기능 사용(멤버 변수). 방향성이 없는 연관은 실선으로, 방향성이 있는 연관은 열린 화살표가 있는 실선으로 표기
	Inheritance / Generalization (상속 / 일반화)	부모 클래스와 자식 클래스 간의 상속 관계. 실선과 비어있는 삼각형을 부모 클래스로 연결
	Realization / Implementation (실현 / 구현)	인터페이스의 함수를 오버라이딩하여 실제 기능으로 구현한 관계. 점선과 비어있는 삼각형을 인터페이스 쪽으로 연결
	Dependency (의존, 종속)	서버와 클라이언트의 관계처럼 한 요소의 정의를 변경하면 다른 요소가 변경될 수 있는 두 요소 사이에 존재함. 지역 변수나 함수의 매개변수로 사용됨. 점선과 열린 화살표로 표기
	Aggregation (집합)	관계의 일부 또는 전체를 나타내는 연관(has). 클래스가 다른 클래스에서 컨테이너 등으로 포함되어 있고, 수명 주기가 다른 클래스(외부 생성, 소멸). 실선과 빈 마름모로 표기
	Composition (구성)	관계의 일부 또는 전체를 나타내는 연관(own). 클래스가 다른 클래스에서 컨테이너 등으로 포함되어 있고, 수명 주기가 같은 클래스(내부 생성, 소멸). 실선과 채워진 마름모로 표기

2. 구현하려는 자신의 기획 사례

플레이어는 자원과 건축물에 대한 정보를 가지고 있으며, 탐험과 건설을 할 수 있으며 매 턴 작업이 완료되었는지 확인할 수 있어야 한다. 또한, 매 턴 건물에서 자원이 생산되는지 확인해야 하며 자원이 생산될 수 있는 경우 해당하는 자원을 획득한다. NPC 와의 거래를 할 때 해당 자원이 0이 아닌지, 거래가능한 자원 양인지 확인 후 거래가 가능해야 한다. 플레이어와 건물들은 화면에 실제로 그려져야 한다.

3. 각 구성 요소별 예시 다이어그램 작성(본인의 기획서 내용 기준)



4. 코드 작성

```
// BasicHeader.h
#pragma once

struct Position
{
    int x;
    int y;
    int z;
};

#include <vector>
#include <map>
#include <string>

using namespace std;
```

```
// cCharacter.h
#pragma once
#include "BasicHeader.h"

class cCharacter
{
public:
    cCharacter();
    virtual ~cCharacter();
    virtual void SetUp() = 0;
    virtual void Update() = 0;
    virtual void Render() = 0;
};
```

```
// cPlayer.h
#pragma once
#include "cResource.h"
#include "cBuilding.h"
#include "cCharacter.h"

class cPlayer : public cCharacter
{
private:
    cResource *m_resource = cResource::GetInstance();
    vector<cBuilding> m_vecBuilding;
    cPlayer();

public:
    ~cPlayer();
    static cPlayer* GetInstance();
    void SetUp();
    void Update();
    void Render();
    bool Explor(Position position);
    bool Build(Position position, string type);
};
```

```

// cBuilding.h
#pragma once
#include "BasicHeader.h"

class cBuilding
{
private:
    string typeName;
    Position pos;
    int gainDate;
    int status;

public:
    cBuilding();
    ~cBuilding();
    void Update();
    void Render();
    bool CheckGainResource();
};

```

```

// cResource.h
#pragma once
#include "BasicHeader.h"

class cResource
{
private:
    int gold;
    int sheep;
    int wood;
    int brick;
    int flour;
    map<string, int> preTrade;

    cResource();

public:
    static cResource* GetInstance();
    ~cResource();
    void SetUp();
    void Update();
    void AddResource(string name, int value);
    void MinusResource(string name, int value);
    bool CheckResourceZero(string name);
    bool CheckCanTrade(string name, int price);
};

```