

# 알고리즘 실습 과제

## 마법사와 몬스터

학번: 2401110252

이름: 박지수

### 보고서 구성

- 1. 개요 ..... 1
  - 가) 요구 사양 ..... 2
  - 나) 게임 진행 및 코드 구조 ..... 3
  - 다) 코드 설명 ..... 4
- 2. 코드 ..... 5
- 3. 결과 스크린샷 ..... 6
- 4. 고찰 ..... 7

## 1. 개요

### 가) 요구 사양

자바의 Scanner 클래스를 사용하여 유저의 입력을 받아서 게임을 진행하고 유저의 명령을 입력받을 개체인 마법사와 그를 상대할 몬스터를 구현하여 입력을 통해 전투를 이어나가야 한다.

특이점은 마법사의 마법 공격은 자원을 소모하고 이때 Random 클래스를 이용하여 30% 확률로 크리티컬이 발동하면 +10의 추가 데미지를 부여한다.

추가로 마법사가 선제공격을 하여 동시에 죽을 일은 없다.

기본 공격과 마법 공격을 사용하여 몬스터를 물리치거나 몬스터의 체력을 전부 소진 시키면 게임이 종료된다.

## 나) 게임 진행 및 코드 구조

### 게임 진행

처음 실행하면 유저의 이름을 정할지 말지 정하는 선택지가 주어진다.

"1" 입력하게 되면 이름을 입력받을 수 있게 되는데

최대 7글자까지 받을 수 있다.

"1" 대신 "2"를 입력하면 기본값 이름(Gandalf)이 저장되고

앞으로 이 이름으로 계속 출력된다.

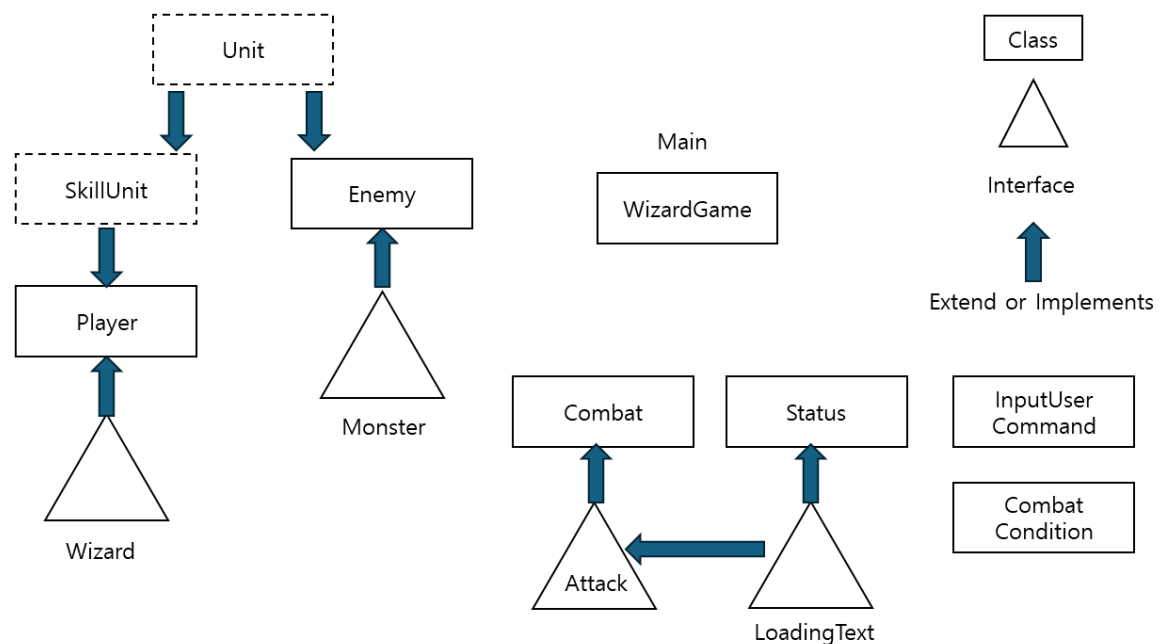
이름까지 기입하고 나면 곧바로 몬스터와 조우하여 전투를 하게 되는데

숫자 "1"또는 "2"를 통해서 기본 공격과 마법 공격을 선택 할 수 있고

숫자 "2"를 입력하고 "1"을 입력하면 등록된 마법 공격을 사용 할 수 있다.

공격을 마치면 적의 공격이 들이닥치고 둘 중 하나의 체력이 줄어 들 때 까지 진행된다.

### 코드 구조



#### 다) 코드 설명

각 소제목의 형태는 전부 class 또는 interface에 해당한다.

#### ㄱ ) WizardGame / Class

main이 포함된 클래스이며 각 클래스들을 실체화 시켜 객체에 접근하여 함수를 실행시킨다.

유저가 조종할 player 그를 상대 할 enemy를 만들고 해당하는 입력 값을 세팅하는 함수도 실행시킨다.

그다음으로는 작명 여부를 결정하고 전투를 반복 하는 while문을 통해 계속 서로의 공격이 오간다. 실행 중 if 조건문을 통해 체력이 0 이하로 떨어졌는지 검사를 하여 break이 실행 될지 말지를 결정한다.

공격을 주고받으면 체력, 데미지 내용이 출력되고 while문 처음으로 돌아간다.

#### ㄴ ) Unit / Class

추상화된 클래스이며 각 Unit에 들어갈 필수적인 요소들만 멤버 변수에 넣었다.

Unit의 직업(종류), 이름, 최대 체력, 체력, 체력 변화가 생기기 전 체력, 기본 공격 데미지, 할 수 있는 목록이 들어가 있다.

#### ㄷ ) SkillUnit / Class

이 클래스 또한 추상 클래스이며 Unit을 상속받고 있다.

SkillUnit이라면 가져야 할 내용들이 들어가 있고 함수로는 가지고 있는 마법의 최소 기력 소모량을 구하는 함수가 있다.

ㄹ ) Player / Class

SkillUnit을 상속받으며 Wizard라는 인터페이스를 합성한다.

이것 말고는 Player 클래스에 다른 내용이 없다.

ㅁ ) Enemy / Class

Unit을 상속받으며 Monster라는 인터페이스를 합성한다.

이 외에는 Player와 동일하다.

ㅂ ) Wizard, Monster / Interface

상숫값과 Set 함수가 들어가 있다.

이 함수는 Player 또는 Enemy 클래스 변수값을 interface의 상수로 바꾸어 준다.

ㅅ ) Combat / Class

이 클래스는 Attack interface와 합성하고 있다.

요구 사양에 공격 외 다른 행동이 포함되어 있지 않기 때문에 추가 내용이 없다.

○ ) Attack / Interface

요구 사양을 보면 30% 확률로 크리티컬이 발생해야 해서  
SecureRandom 클래스를 쓰기 위해 Import 하였고  
크리티컬 도중 긴장감을 주기 위해서 LoadingText interface를 합성했다.

30% 확률을 통과했는지 확인하는 Boolean 값을 반환 받기 위해  
CreateRandomValue 함수가 있다.

공격 외 다른 곳에도 확률이 필요하면 따로 인터페이스 또는 클래스를  
추가하여 관리 할 예정이다.

OriginAttack 함수는 Unit의 기본공격을 할 때 쓰는 함수이다.

SkillAttack 함수는 Unit중 SkillUnit이 쓰는 함수이다.

이 함수에서는 따로 MP에 관한 조건 검사를 실행하지 않고  
MP를 소모시키고 if 조건문에 확률을 뚫었으면 추가 데미지가 들어간다.

UnitAttack 함수는 위에 상기한 함수들을 포함한 함수이다.

파라미터들을 살펴보면 숫자와, 공격하는 Unit, 맞는 입장의 Unit,  
공격 과정을 출력하기 위한 Status, Skill을 사용할 경우 추가 조작이 필요해  
InputUserCommand까지 필요하다.

배열로 행동 가능범위를 지정했기 때문에 입력 받은 번호를 -1 한다.  
그리고 switch문으로 무엇을 선택하였는지 받는데

1번인 경우 OriginAttack 함수로 넘어가고  
2번인 경우 SkillAttack 함수쪽으로 넘어간다.

2번을 선택하면 1번보다 살짝 복잡하게 진행된다.  
먼저 공격 하는 Unit의 Class가 Player인지 검사한다.  
Player라면 Unit 클래스를 Player 클래스로 바꾸고  
그에 대한 멤버 및, 함수에 접근한다.

만약 MP가 부족해 못쓰는 상황이지만 쓰고 싶다고 입력 받는 상황이 나올 수 있어 두가지 정도로 나뉘었다.

보유 하고 있는 Skill의 MP소모량을 다 조사한 MinSkillCost 보다 현재 MP가 적을 경우 기본공격이 바로 실행 되고 만약 스킬이 여러 개 있을 경우를 생각해 SelectSkill 에서 다시 입력한 스킬 종류를 다시 현재 MP에 적합한지 검사하고 아닌 경우 상황에 맞게 입력을 받는다.

#### ㄱ ) CombatCondition / Class

전투하는 도중 특별한 상황에 대해 관련 함수를 만들고자 하였지만 지금은 둘 중 하나가 죽는 경우 밖에 없다고 생각해 하나가 죽었을 시 main의 while문을 탈출하게 했다.

EscapeCondition 함수에서 Unit의 HP가 0 이하가 될 경우 false를 반환해서 IsEscape 함수를 도와준다.

IsEscape 함수는 위의 EscapeCondition을 포함한 함수로 죽으면 관련 print문이 출력되고 false를 반환해 main의 while문을 탈출한다.

죽지 않는다면 print문이 출력 되고 true를 반환해 탈출하지 못하게 한다.

## 大 ) Status / Class

Status 클래스는 게임의 진행 상황을 알려주는 클래스이다.

TimeSleep 함수는 텍스트를 반복적으로 출력 하고 싶을 때 쓸려고 만들었다.

MeetingScene 함수에서 사용하고 있고

게임이 시작될 때 경고문을 출력하기 위해 만들었다.

UnitStatus 함수는 먼저 Unit 클래스의 객체를 입력받고

SkillUnit 클래스 객체인지 확인을 한다

확인하는 이유는 SkillUnit일 경우 MP까지 표시하고 Unit인 경우는 MP가 없기 때문이다.

SelectText 함수는 Scanner 클래스를 통해 유저가 입력을 할 때

안내문을 보아야 입력 할 것이다.

바로 그 안내문을 출력해주는 함수이다.

AfterCombatStatusText, Symbol 함수는 서로 한 턱 씩 공격한 후

HP에 관한 내용을 정리하는 함수이다.

Symbol은 이모티콘이 출력이 된다고 가정하고 만든 함수이고

안된다면 Text로 함수를 바꿔서 출력하면된다.

AfterUnitSkillStatus 함수는

사용한 Skill과 SkillCost, 현재 MP량을 출력 해주는 함수이다.

EndCombatPrint 함수는 누구 하나가 죽었을 때 발생하는 출력문으로

Player의 객체가 죽었을 경우 랜덤한 나이가 출력되고

Enemy의 객체가 죽었을 경우 객체의 이름과 함께 쓰러뜨렸다고 출력된다.

NotEnoughMPPrint 함수는 현재 MP보다 높은 Skill을 사용할 때 나오는

경고문이다.



UnitHPPrint 함수는 위의 Symbol 함수의 내부에 있던 것으로 어떤 파라미터를 받는지에 따라 다르게 작동되도록 만들었다.

Unit 클래스의 객체를 받으면 체력을 표시해주고  
int 자료형으로 받으면 damage의 크기를 이모티콘으로 출력해준다.

⇒ ) InputUserCommand / Class

객체가 생성됨과 동시에 Scanner 클래스 객체도 생성시킨다.  
주로 입력과 관계된 클래스이다.

InputArray 함수는 공격 또는 마법과 같은 선택지를 전부 배열로 구성하였는데 선택지에 없는 값을 입력하면 안되니 해당하는 배열로 받고 그에 맞게 선택지를 출력 및 조건을 걸기 위해서 배열로 받게 했다.

ParsingInt 함수는 InputArray 안에서 작동하는 함수로 try catch를 통해 숫자가 아닌 값을 입력 받았을 때 다시 받게끔 하는 함수이다. 도중에 숫자이지만 입력 받은 배열의 범위를 벗어난다면 범위 안쪽을 입력하게끔 도와준다.

SelectSkill 함수는 Skill을 정할 때 사용하는 함수이다.  
마찬가지로 InputArray 함수를 사용하여 만들었다.  
다른 점은 MP검사도 하고 MP관련 출력문도 발생시킨다.

SetName 함수는 Unit의 이름을 정할 때 사용한다.  
게임의 처음에 작명 할 것인지 묻는데 이때 작동하는데 최대 숫자는 문자열 길이로 체크해서 검사한다.

꼭 Int로 받을 필요도 없어서 따로 try catch 하지 않았다.

## ㅁ ) LoadingText / Interface

따로 Sleep과 특수문자들을 활용해 크리티컬을 발생시키는  
Skill 공격을 할 때 조금 더 긴장감을 주기 위함과

게임을 처음 구동 했을 때 심심한 느낌이 들어 로딩 창 느낌으로 만들고 싶어  
추가로 interface를 마지막에 만들었다.

LoadingTextPrint 함수는 랜덤을 발생시켜 별의 개수를 조정하고 출력한다.

계속 별의 개수가 똑같으면 재미가 없을 것 같아 일정 범위를 두었고  
같은별이 계속 출력되면 그것 또한 재미가 없어 나머지 연산자로 번갈아  
출력하게 만들었다.

GameLoadingText 함수는 게임을 처음 구동하였을 때  
나오는 로딩 창과 같은 함수이다.

이부분도 재생할 때 마다 조금 더 다름을 주고 싶어 랜덤을 넣었다.

## 2. 코드

```
// 2024_04_16_알고리즘_실습과제
// 한국폴리텍대학_서울정수캠퍼스_인공지능소프트웨어학과
// 2401110252_박지수
// 마법사로 몬스터를 물리치는 게임
// https://github.com/Mourn5367/WizardVer2

public class WizardGame
{
    public static void main(String[] args) throws InterruptedException {
        InputUserCommand inputUserCommand = new InputUserCommand();
        CombatCondition combatCondition = new CombatCondition();
        Status status = new Status();
        Combat combat = new Combat();
        Player player = new Player();
        Enemy enemy = new Enemy();

        player.SetWizard(player); // 유저는 Wizard 적은 Monster 세팅
        enemy.SetMonster(enemy);
        status.GameLoadingText();

        inputUserCommand.SetName(player, status); // 작명 여부

        status.MeetingScene(player, enemy);
    }
}
```

```

        while (true)
        {
            status.SelectText(player.ableList); // 선택지 출력

            combat.UnitAttack(inputUserCommand.InputArray(player.ableList), player, enemy
                , status, inputUserCommand); // 유저 공격
            if(!combatCondition.IsEscape(enemy, status)) {break;}
            // 적이 죽었는지 검사

            status.UnitStatus(enemy); // 적 상태 출력
            combat.UnitAttack(1, enemy, player
                , status, inputUserCommand);
            // 적이 입력을 못받아 기본공격하게 입력설정함

            if(!combatCondition.IsEscape(player, status)) {break;} // 유저가 죽었는지
            검사

            status.UnitStatus(player); // 유저 상태 출력
            status.AfterCombatStatusSymbol(player, enemy); // 전투 요약
        }
    }
}

abstract class Unit
{
    // 모든 유닛은 이 값은 가지고 있어야 함.
    protected String unitClass;
    protected String name;
    protected int maxHP;
    protected int HP;
    protected int beforeHP;
    protected int originDMG;
    protected String[] ableList;
}

abstract class SkillUnit extends Unit
{
    int skillDMG[]; // 스킬 공격 데미지
    int criticalDMG; // 크리티컬 추가 데미지 량
    double criticalPercent; // 크리티컬 계수
    int maxMP; // 최대 MP
    int MP; // 현재 MP
    int beforeMP; // 전투가 일어나기 전 MP
    int skillCost[]; // 스킬 MP 소모량
    String skill[]; // 스킬 이름
    int minCost; // 스킬중 최소 MP 소모량

    int MinSkillCost() // 최소 MP 소모량 구하기
    {
        int minCost = 0;
        for (int i = 0; i < skillCost.length; i++)
        {
            minCost = skillCost[0];
            if (minCost > skillCost[i])
            {
                minCost = skillCost[i];
            }
        }
    }
}

```

```

        return minCost;
    }
}

class Player extends SkillUnit implements Wizard
{
    public Player()
    {

    }
}

class Enemy extends Unit implements Monster
{

}

public interface Wizard
{
    // 직업이 Wizard 인 경우 이 값을 가져야 함
    String WIZARDNAME = "Gandalf";
    String UNITCLASSNAME = "Wizard";
    int WIZARDMAXHP = 80;
    int WIZARDMAXMP = 100;
    String[] WIZARDABLELIST = {"기본 공격", "마법 공격"};
    String[] WIZARDSKILL = {"짱센마법", "더 짱센마법"};
    int[] WIZARDSKILLCOST = {30, 60};
    int WIZARDORIGINDMG = 5;
    int[] WIZARDSKILLDMG = {30, 80};
    int WIZARDCRITICALADDITIONALDMG = 10;
    double WIZARDCRITICALPERCENT = 0.3;
    // 마법 별로 크리티컬이 달라지면 크리티컬 계수도 배열로 바꿔야함
    default void SetWizard(Player _player)
    {
        _player.unitClass = UNITCLASSNAME;
        _player.name = WIZARDNAME;
        _player.maxHP = WIZARDMAXHP;
        _player.HP = _player.maxHP;
        _player.beforeHP = 0;
        _player.originDMG = WIZARDORIGINDMG;
        _player.ableList = WIZARDABLELIST; // 여기 까지 Unit 값 설정

        _player.skillDMG = WIZARDSKILLDMG;
        _player.criticalDMG = WIZARDCRITICALADDITIONALDMG;
        _player.criticalPercent = WIZARDCRITICALPERCENT;

        _player.maxMP = WIZARDMAXMP;
        _player.MP = _player.maxMP;
        _player.beforeMP = 0;

        _player.skillCost = WIZARDSKILLCOST;
        _player.skill = WIZARDSKILL;

        _player.minCost = _player.MinSkillCost();
    }
}

public interface Monster
{
    String MONSTERNAME = "몬스터";

```

```

int MONSTERMAXHP = 100;
int MONSTERORIDMG = 10;

String UNITCLASSNAME = "적";
String MONSTERABLELIST[] = {"홍폭한 일격"};

default void SetMonster(Enemy _enemy)
{
    _enemy.unitClass = UNITCLASSNAME;
    _enemy.name = MONSTERNAME;
    _enemy.maxHP = MONSTERMAXHP;
    _enemy.HP = _enemy.maxHP;
    _enemy.originDMG = MONSTERORIDMG;
    _enemy.ableList = MONSTERABLELIST;
}
}

public class Combat implements Attack
{
}

import java.security.SecureRandom;
interface Attack extends LoadingText
{
    SecureRandom random = new SecureRandom();
    default boolean CreatRandomValue(SkillUnit _skillUnit) // 0 은 크리티컬
    {
        double ranValue = random.nextDouble();
        return ranValue < _skillUnit.criticalPercent; // 확률을 통과 했으면 true
    }

    default void OriginAttack(Unit _offense, Unit _defense) // 기본 공격할 때
    {
        System.out.printf("%s !!\n", _offense.ableList[0]);
        _defense.beforeHP = _defense.HP;
        _defense.HP -= _offense.originDMG;
    }

    default void SkillAttack(SkillUnit _skillUnit, Unit _deffense,
                             int _selectSkill) throws InterruptedException // skill 공격
    함수
    {
        _skillUnit.beforeMP = _skillUnit.MP;
        _skillUnit.MP -= _skillUnit.skillCost[_selectSkill];
        System.out.printf("%s !!\n", _skillUnit.skill[_selectSkill]);
        LoadingTextPrint();

        if (CreatRandomValue(_skillUnit)) // 크리티컬 확률을 통과했으면
        {
            System.out.println("!! 크리티컬 발생 !!");
            _deffense.beforeHP = _deffense.HP;
            _deffense.HP -= _skillUnit.skillDMG[_selectSkill] +
            _skillUnit.criticalDMG;
        }
        else // 통과 못한 경우
        {
            _deffense.beforeHP = _deffense.HP;
            _deffense.HP -= _skillUnit.skillDMG[_selectSkill];
        }
    }

    default void UnitAttack

```

```

        (int _selectAttack, Unit _unit, Unit _defense, Status _status,
         InputUserCommand _inputUserCommand) throws InterruptedException
    {
        // 유닛이 공격을 할때 기본 공격 할지 마법 공격할지 이 함수뒤로 나뉘어진다.
        _selectAttack -= 1; // 배열은 0 번 부터라 1 차감
        Player player;
        Enemy enemy;
        switch (_selectAttack) // 1. 기본 공격 2. 마법공격
        {
            case 0:// 기본공격은 1 번
                OriginAttack(_unit,_defense);
                break;
            case 1:// 마법공격은 2 번
                if(_unit instanceof Player )
                {
                    player = (Player) _unit;
                    if (player.MP < player.MinSkillCost())
// 최소 마나 마법보다 적은 MP 일 경우
                {
                    System.out.printf("기력이 최소 요구량 %d 보다 작아 %s(을)를
사용합니다.\n",
                                player.MinSkillCost(), player.ableList[0]);
                    OriginAttack(player,_defense); // 기본공격을 하게 설정
                }
                else // 아니라면 마법을 선택한다.
                {
                    int selectSkill = 0; // 마법 선택을 담을 변수
                    _status.SelectText(player.skill);
                    selectSkill = _inputUserCommand.SelectSkill(player, _status);
// 입력받기

                    SkillAttack(player, _defense, selectSkill);
                    _status.AfterUnitSkillStatus(player,selectSkill);
                }
                break;
            }
            else if(_unit instanceof Enemy)
                // 스킬을 사용하는 Enemy 의 경우는 입력 받을 수 없으니 따로 클래스 검사해서
예외처리

                // 아직은 스킬을 사용하는 Enemy 가 없다.
                {
                }
                break;
        }
    }
}

public class CombatCondition
{
    public boolean EscapeCondition(Unit _unit)
    {
        // 유닛의 체력이 0 이하 이면 true 반환
        return _unit.HP <= 0;
    }
    public boolean IsEscape(Unit _unit, Status status) throws InterruptedException {
        Thread.sleep(1000);
        if (EscapeCondition(_unit)) // true 일 경우
    }
}

```

```

    {
        status.EndCombatPrint(EscapeCondition(_unit), _unit);
        Thread.sleep(500);
        return false; // false 를 반환해 while 문 종료에 도움을 줌
    } else {
        System.out.printf("공격을 마쳤다...\n");
        Thread.sleep(500);
        return true;
    }
}
}

```

```

import java.util.Random;
public class Status implements LoadingText
{
    Random random = new Random();

    public void TimeSleep(int _time, String _message) // 3 번 멈추는 함수
    {
        for (int i = 0; i < 3; i++ )
        {
            System.out.printf("%s", _message);
            try {Thread.sleep(_time);}
            catch (InterruptedException InterException)
            {InterException.printStackTrace();}
        }
    }

    public void MeetingScene(Unit _unit1, Unit _unit2)
    {
        TimeSleep(500, "몬스터 접근중 !\n");

        System.out.printf("%s %s(이)가 야생의 %s %s를 만났다!!\n\n"
            , _unit1.unitClass, _unit1.name, _unit2.unitClass, _unit2.name);
    }

    public void UnitStatus(Unit _unit) // 스킬 유닛일 경우 MP 표시 아니면 HP 까지만
    {
        SkillUnit skillUnit;
        if (_unit instanceof SkillUnit)
        {
            skillUnit =(SkillUnit) _unit;
            System.out.printf("%s: %s 체력: %d/%d 기력: %d/%d\n"
                , skillUnit.unitClass, skillUnit.name, skillUnit.HP, skillUnit.maxHP,
                skillUnit.MP, skillUnit.maxMP);
        }
        else
        {
            System.out.printf("%s: %s 체력: %d/%d\n"
                , _unit.unitClass, _unit.name, _unit.HP, _unit.maxHP);
        }
    }

    public void SelectText(String[] _array) // 선택지 출력
    {
        for (int i = 0; i < _array.length; i++)
        {
            System.out.printf("%d. %s\t", i+1, _array[i]);
        }
        System.out.println();
    }

    public void AfterCombatStatusText(Unit _unit1, Unit _unit2) //이모티콘이 깨질경우
    {

```

```

        System.out.printf("%s(이)가 %d의 피해를 입어 %d/%d의 체력이 되었고 %s(이)가 %d의
피해를 입어 %d/%d 체력이 되었다!\n"
        , _unit1.name, _unit1.beforeHP - _unit1.HP, _unit1.HP, _unit1.maxHP,
        _unit2.name, _unit2.beforeHP - _unit2.HP, _unit2.HP, _unit2.maxHP);
    }
    public void AfterCombatStatusSymbol(Unit _unit1, Unit _unit2)
// 전투요약을 이모티콘으로
    {
        System.out.printf("%s(이)가 ", _unit1.name);
        UnitHPPrint(_unit1.beforeHP - _unit1.HP);
        System.out.print("의 피해를 입어 ");
        UnitHPPrint(_unit1);
        System.out.printf("의 체력이 되었고 %s가 ", _unit2.name);
        UnitHPPrint(_unit2.beforeHP - _unit2.HP);
        System.out.print("의 피해를 입어 ");
        UnitHPPrint(_unit2);
        System.out.println("의 체력이 되었다...!!\n");
    }
    public void AfterUnitSkillStatus(SkillUnit _skillUnit, int selectSkill)
    {
        System.out.printf("%s을(를) 사용하여 기력을 %d 사용했다\n" +
            "기력이 %d/%d 남았다 !\n", _skillUnit.skill[selectSkill],
            _skillUnit.skillCost[selectSkill],
            _skillUnit.MP, _skillUnit.maxMP);
    }
    public void EndCombatPrint(boolean _isEscapeCombatCondition, Unit _unit)
        // 전투가 결착이 났을때 나오는 함수
    {
        if (_isEscapeCombatCondition)
        {
            if (_unit instanceof Player)
            {
                UnitStatus(_unit);
                System.out.printf("%s는 %d의 나이로 생을 마감했다...\n", _unit.name,
                    random.nextInt(20,100));
            }
            else if (_unit instanceof Enemy)
            {
                UnitStatus(_unit);
                System.out.printf("%s를 쓰러뜨렸다...!!!\n", _unit.name);
            }
        }
    }
    public void NotEnoughMPPrint(SkillUnit skillUnit, int _selectSkill)
        // 선택한 스킬보다 MP가 없을 경우
    {
        System.out.printf("기력이 %d이므로 기력 %d(을)를 소모하는 %s을 할수없어 !!\n"
            , skillUnit.MP, skillUnit.skillCost[_selectSkill], skillUnit.skill[_selec
tSkill]);
    }

    public void UnitHPPrint(Unit _unit)
// 현재 DMG 최소 단위가 5부터 하고 있어 10 아래의 체력은 깨진하트
// 10부터는 완성된 하트 줄어든 체력은 빈 하트로 표현
    {
        if (_unit.HP % 10 !=0)

```



```

        {
            System.out.printf("♡");
        }
        for (int i = 0; i < _unit.HP/10; i++)
        {
            System.out.printf("♡");
        }
        for (int i = 0; i < (_unit.maxHP - _unit.HP)/10; i++)
        {
            System.out.printf("♡");
        }
    }
    public void UnitHPPrint(int _damage)
    {
        if (_damage % 10 != 0)
        {
            System.out.printf("♡");
        }
        else if (_damage <= 0)
        {
            System.out.printf("♡");
        }
        for (int i = 0; i < _damage/10; i++)
        {
            System.out.printf("♡");
        }
    }
}

import java.util.Scanner;

public class InputUserCommand
{
    public Scanner scanner;
    public InputUserCommand()
    {
        this.scanner = new Scanner(System.in);
    } // 객체가 생성될때 스캐너 객체 생성

    public int InputArray(String[] _inputArray)
    {
        return ParsingInt(_inputArray);
    }
    // 행동 선택지를 넣고 유저가 숫자 입력값을 반환

    public int ParsingInt(String[] _inputArray )
    // 선택지 이외의 숫자, 숫자 이외의 값을 거르기 위해
    {
        boolean noneException = false;
        int noneExceptionIntText = 0;
        while(!noneException)
        {
            try
            {
                noneExceptionIntText = Integer.parseInt(scanner.nextLine());
                if (noneExceptionIntText > 0 && noneExceptionIntText <=
_inputArray.length )
                    //여기서 선택지를 이외의 값을 가져오면 다시 받게하자
                {
                    noneException = true;
                }
            }
            else

```

```

        {
            System.out.print("나는 ");
            for (int i = 0; i < _inputArray.length; i++)
            {
                System.out.printf("숫자 %d\t", i+1);
            }

            System.out.print("말고는 못 알아먹겠어 !\n");
            noneException = false;
        }
    }
    catch (NumberFormatException numException)
    {
        System.out.print("나는 ");
        for (int i = 0; i < _inputArray.length; i++)
        {
            System.out.printf("숫자 %d\t", i+1);
        }

        System.out.print("말고는 못 알아먹겠어 !\n");
    }
}
return noneExceptionIntText;
}

public int SelectSkill(SkillUnit _skillUnit, Status _status)
// 어떤 스킬을 고를지 입력받음
{
    int selectSkill = InputArray(_skillUnit.skill) - 1;

    while(_skillUnit.MP < _skillUnit.skillCost[selectSkill])
    {
        _status.NotEnoughMPPrint(_skillUnit, selectSkill);
        _status.SelectText(_skillUnit.skill);
        selectSkill = InputArray(_skillUnit.skill) - 1;
    }
    return selectSkill;
}

public void SetName(Unit _unit, Status status) // 유닛을 작명함
{
    int maxNameLength = 7;
    String setNameMenu[] = {"작명한다.", "기본이름으로"};
    status.SelectText(setNameMenu);
    int inputMenu = ParsingInt(setNameMenu);
    if (inputMenu == 1)
    {
        System.out.print("이름을 입력하세요 최대 7 글자까지 됩니다. :");
        while(true)
        {
            String unitName = scanner.nextLine();
            if (unitName.length() > maxNameLength)
            {
                System.out.println("7 글자를 초과하였습니다.");
                System.out.print("이름을 입력하세요 최대 7 글자까지 됩니다. :");
            }
            else
            {
                _unit.name = unitName;
            }
        }
    }
}

```

```

        break;
    }
}
}
System.out.printf("\n 당신의 이름은 %s 입니다 !\n\n", _unit.name);
}
}
import java.util.Random;
public interface LoadingText {
    Random random = new Random();
    default void LoadingTextPrint() throws InterruptedException
        // 확률적인 일이 생길때 넣을 로딩 텍스트
        // 현재 마법공격에만 있어 그곳에만 넣고있음
    {
        int loadingText = random.nextInt(4, 8);
        for (int i = 1; i < loadingText; i++)
        {
            if ( i % 2 == 0)
            {
                System.out.print("☆");
                Thread.sleep(500);
            }
            else
            {
                System.out.print("★");
                Thread.sleep(500);
            }
        }
        System.out.println();
    }
}

default void GameLoadingText() throws InterruptedException
// 게임 구동 시 실행될 로딩 창
{
    int loadingText = random.nextInt(2, 4);

    System.out.print("마법사와 몬스터 구동중");
    for (int i = 0; i < 3; i++)
    {
        System.out.print(". ");
        Thread.sleep(1000);
    }
    System.out.print("\n");
    System.out.print("준비가 거의 다 되었습니다.\n");
    System.out.print("[");
    for (int i = 0; i < loadingText; i++)
    {
        System.out.print("|");
        Thread.sleep(random.nextInt(1000, 2000));
    }
    for (int i = 0; i < 3; i++)
    {
        System.out.print("|");
        Thread.sleep(300);
    }
    System.out.print("]\n");
    System.out.println("준비되었습니다.");
    Thread.sleep(1000);
}
}

```

### 3. 결과 스크린샷

```
마법사와 몬스터 구동중. . .
준비가 거의 다 되었습니다.
[|||||]
준비되었습니다.
1. 작명한다. 2. 기본이름으로
2

당신의 이름은 Gandalf입니다 !

몬스터 접근중 !
몬스터 접근중 !
몬스터 접근중 !
Wizard Gandalf(이)가 야생의 적 몬스터를 만났다!!

1. 기본 공격 2. 마법 공격
2
1. 팡센마법 2. 더 팡센마법
1
팡센마법 !!
☆☆☆
팡센마법을(를) 사용하여 기력을 30 사용했다
기력이 70/100 남았다 !
공격을 마쳤다...
적: 몬스터 체력: 70/100
훌륭한 일격 !!
공격을 마쳤다...
Wizard: Gandalf 체력: 70/80 기력: 70/100
Gandalf(이)가 ❤️의 피해를 입어 ❤️❤️❤️❤️❤️❤️❤️❤️의 체력이 되었고 몬스터가 ❤️❤️❤️의 피해를 입어 ❤️❤️❤️❤️❤️❤️❤️❤️❤️❤️❤️의 체력이 되었다...!!

1. 기본 공격 2. 마법 공격
2
1. 팡센마법 2. 더 팡센마법
1
팡센마법 !!
☆☆☆
!! 크리티컬 발생 !!
팡센마법을(를) 사용하여 기력을 30 사용했다
기력이 40/100 남았다 !
공격을 마쳤다...
적: 몬스터 체력: 30/100
훌륭한 일격 !!
공격을 마쳤다...
Wizard: Gandalf 체력: 60/80 기력: 40/100
Gandalf(이)가 ❤️의 피해를 입어 ❤️❤️❤️❤️❤️❤️❤️❤️의 체력이 되었고 몬스터가 ❤️❤️❤️의 피해를 입어 ❤️❤️❤️❤️❤️❤️❤️❤️❤️❤️❤️의 체력이 되었다...!!

1. 기본 공격 2. 마법 공격
2
1. 팡센마법 2. 더 팡센마법
1
팡센마법 !!
☆☆☆
!! 크리티컬 발생 !!
팡센마법을(를) 사용하여 기력을 30 사용했다
기력이 10/100 남았다 !
적: 몬스터 체력: -10/100
몬스터를 쓰러뜨렸다...!!!

Process finished with exit code 0
```

마법 공격만 사용 하여 쓰러뜨린 경우

마법사와 몬스터 구동중. . .

준비가 거의 다 되었습니다.

[|||||]

준비되었습니다.

1. 작명한다. 2. 기본이름으로

2

당신의 이름은 Gandalf입니다 !

몬스터 접근중 !

몬스터 접근중 !

몬스터 접근중 !

Wizard Gandalf(이)가 야생의 적 몬스터를 만났다!!

1. 기본 공격 2. 마법 공격

2

1. 광선마법 2. 더 광선마법

1

광선마법 !!

☆☆☆

광선마법을(를) 사용하여 기력을 30 사용했다

기력이 70/100 남았다 !

공격을 마쳤다...

적: 몬스터 체력: 70/100

흠뻑한 일격 !!

공격을 마쳤다...

Wizard: Gandalf 체력: 70/80 기력: 70/100

Gandalf(이)가 ♥의 피해를 입어 ♥♥♥♥♥♥♥♥♥♥♥♥의 체력이 되었고 몬스터가 ♥♥♥♥의 피해를 입어 ♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥의 체력이 되었다...!!

1. 기본 공격 2. 마법 공격

2

1. 광선마법 2. 더 광선마법

1

광선마법 !!

☆☆☆

광선마법을(를) 사용하여 기력을 30 사용했다

기력이 40/100 남았다 !

공격을 마쳤다...

적: 몬스터 체력: 40/100

흠뻑한 일격 !!

공격을 마쳤다...

Wizard: Gandalf 체력: 60/80 기력: 40/100

Gandalf(이)가 ♥의 피해를 입어 ♥♥♥♥♥♥♥♥♥♥♥♥♥♥의 체력이 되었고 몬스터가 ♥♥♥♥의 피해를 입어 ♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥의 체력이 되었다...!!

1. 기본 공격 2. 마법 공격

2

1. 광선마법 2. 더 광선마법

1

광선마법 !!

☆☆☆

광선마법을(를) 사용하여 기력을 30 사용했다

기력이 10/100 남았다 !

공격을 마쳤다...

적: 몬스터 체력: 10/100

흠뻑한 일격 !!

공격을 마쳤다...

Wizard: Gandalf 체력: 50/80 기력: 10/100

Gandalf(이)가 ♥의 피해를 입어 ♥♥♥♥♥♥♥♥♥♥♥♥♥♥의 체력이 되었고 몬스터가 ♥♥♥♥의 피해를 입어 ♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥의 체력이 되었다...!!

1. 기본 공격 2. 마법 공격

2

기력이 최소 요구량 30 보다 작아 기본 공격(를)을 사용합니다.

기본 공격 !!
































공격을 마쳤다...

적: 몬스터 체력: 5/100

흠뻑한 일격 !!

공격을 마쳤다...

Wizard: GandaIf 체력: 40/80 기력: 10/100

GandaIf(이)가 의 피해를 입어 의 체력이 되었고 몬스터가 의 피해를 입어 

1. 기본 공격 2. 마법 공격

1

기본 공격 !!

공격을 마쳤다...

적: 몬스터 체력: 75/100

훌륭한 일격 !!

공격을 마쳤다...

Wizard: 박지수 체력: 30/80 기력: 100/100

박지수(이)가 ❤️의 피해를 입어 ❤️❤️❤️❤️❤️❤️❤️의 체력이 되었고 몬스터가 ❤️의 피해를 입어 ❤️❤️❤️❤️❤️❤️❤️❤️❤️❤️의 체력이 되었다...!!

1. 기본 공격 2. 마법 공격

1

기본 공격 !!

공격을 마쳤다...

적: 몬스터 체력: 70/100

훌륭한 일격 !!

공격을 마쳤다...

Wizard: 박지수 체력: 20/80 기력: 100/100

박지수(이)가 ❤️의 피해를 입어 ❤️❤️❤️❤️❤️❤️❤️의 체력이 되었고 몬스터가 ❤️의 피해를 입어 ❤️❤️❤️❤️❤️❤️❤️❤️❤️❤️의 체력이 되었다...!!

1. 기본 공격 2. 마법 공격

1

기본 공격 !!

공격을 마쳤다...

적: 몬스터 체력: 65/100

훌륭한 일격 !!

공격을 마쳤다...

Wizard: 박지수 체력: 10/80 기력: 100/100

박지수(이)가 ❤️의 피해를 입어 ❤️❤️❤️❤️❤️❤️❤️의 체력이 되었고 몬스터가 ❤️의 피해를 입어 ❤️❤️❤️❤️❤️❤️❤️❤️❤️❤️의 체력이 되었다...!!

1. 기본 공격 2. 마법 공격

1

기본 공격 !!

공격을 마쳤다...

적: 몬스터 체력: 60/100

훌륭한 일격 !!

Wizard: 박지수 체력: 0/80 기력: 100/100

박지수는 50의 나이로 생을 마감했다...

유저가 죽는 경우

#### 4. 고찰

두 번째 과제인 만큼 내가 조금 더 성숙한 모습으로 과제를 풀어나가길 원했다.

그렇게 하기 위해서는 지난번 과제에서 했던 잘못, 실수를 줄이기 위해 제일 큰 패착이라고 생각했던 코드부터 작성하기 보다는 작성 전 구조를 설계하고 어떻게 작성할 것인지 생각부터 했다.

그 다음 생각을 텍스트와 그림을 곁들여서 기획서를 쓴다는 느낌으로 정리하고 그 내용을 참고하면서 코드를 작성해 나갔다.

아직 배우는 과정이고 하니 완벽하게 구조를 설계하고 할 수는 없었지만 생각을 해봤다는 것과 그 생각을 글과 그림으로 그렸다는 것이 구조를 짤 때 큰 힘이 되었던 것 같다.

자바 수업시간때는 멤버 변수와 클래스 내 함수 작성, main으로 가져와서 실체화만 했을 것이다.

그것 가지고는 이번 과제를 하기에 부족하다고 느낄 찰나 유튜브 알고리즘에 SOLID 원칙을 소개하는 영상이 있었다. 덕분에 내가 지금 작성한 것이 원칙에 아주 위배되고 있다는 것을 자각하고 이를 고치기 위해서 주말동안 상속과 합성을 알아보았는데 지금 과제에서 적용하기 너무 좋은 상황이었다.

공부하기 전에는 Player 클래스와 Enemy 클래스, 수치를 입력하는 Value 클래스 이렇게 나뉘었는데 Value 클래스를 없애고 Unit이라는 추상 클래스를 만들었다.

설계를 Unit이라면 무조건 가지고 있을 체력, 기본 공격, 이름 정도만 Unit 클래스에 넣고 SkillUnit에는 Skill과 관련된 MP, Skill 목록, 크리티컬 계수등 Skill과 관련된 멤버를 넣었다.

그리고 Wizard Interface에는 Wizard에 해당하는 값을 전부 넣었다. Wizard의 HP, MP, DMG, Skill등 다 세팅할 수 있게 함수도 넣었다.

이는 Monster 클래스도 마찬가지다.



다음으로 크게 바꾼 것이 공격 관련 내용이다.

기존에는 Combat 클래스 안에 Player가 기본공격, 마법공격 하는 함수 만들고 Enemy가 공격하는 함수 이렇게 만들었지만,

생각해보니 이러면 Enemy가 새로 생기거나 Wizard가 아니라 Knight 이렇게 새로운 클래스가 생기면 다 하나씩 만들 생각을 하니 이건 매우 잘못되었다 생각 해

Unit으로 파라미터를 받았다.

싸움을 하는 객체는 유닛임에는 틀림없어서 모든 객체가 들어가서 공격을 할 수 있게 다시 재구성하였다.

그리고 Combat Class는 공격외에 회피, 도망, 소모품 사용하는 것 처럼 추가 될 사항은 새로운 interface를 만들어서 합치면 Combat을 타고 들어가면 되니

Unit쪽에 새롭게 멤버나 함수 추가하고 관련 interface를 만들면 되니 크게 수정 할 것이 없다.

기존처럼 공격을 하나씩 만들었으면 추가 될 사항도 하나씩 만들어야 하는데 작업량이 정말 말도 안된다.

이번 과제를 하면서 어떨 때 상속을 하고 interface를 만들어 합성해야 할 지 경험이 생겼고 함수 하나, 변수 하나 만들 때 마다 꼭 필요한지,

재사용성이 높게 작성을 하였는지 신중히 검토하고 설계해야 한다는 뜻 깊은 교훈을 얻었다.