알고리즘 실습 과제
마법사와 몬스터

학번: 2401110252

이름: 박지수

보고서 구성

1. 개요

가) 요구 사양

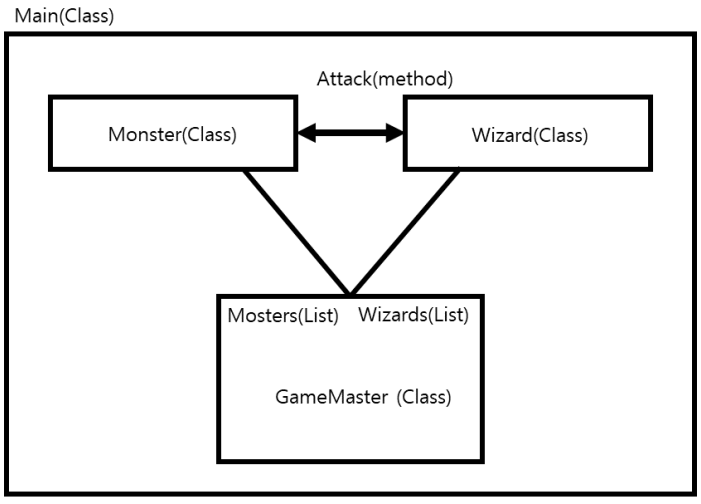
마법사와 몬스터 각각의 클래스를 구분 지어 코드를 작성한다. 작성한 클래스를 바탕으로 객체를 실체화한 다음 함수를 실행하여 객체의 속성을 수정 및 확인을 하고 객체끼리 상호작용을 한다.

상호작용의 내용은 일반 공격과 마법 공격이 있으며 서로의 HP가 다할 때까지 진행된다.

나) 게임 진행 및 코드 구조

게임 진행은 마법사, 몬스터의 개체 수를 각각 사용자로부터 입력받은 다음 전투를 시작한다.

전투의 순서는 마법사(플레이어)부터 시작하게 되며 플레이어는 행동할 마법사를 지정하고 공격한다. 한번 행위를 한 마법사는 다시 행동할 수 없게 선택 창이 나타나지 않으며 모든 마법사가 공격한 후에는 몬스터의 공격 차례로 넘어간다. 몬스터는 무작위의 마법사를 공격하게 된다.
몬스터의 숫자가 0 혹은 마법사의 숫자가 0이 된다면 게임은 그대로 종료된다.



Main 클래스 안에서 전체적인 코드가 진행되어 있고 Monster, Wizard클래스는 GameMaster 클래스 내의 Monsters, Wizards 리스트를 통해서 관리하고 생성

한다. 그리고 두 객체는 Attack 매서드를 사용하여 상호작용을 한다.

다) 코드 설명

클래스는 Wizard, Monster, GameMaster, Main 총 4개의 클래스로 구성되어 있으며 주요 코드를 설명합니다.

다-ㄱ) Wizard/Class

SelectMonster

마법사가 어떠한 몬스터를 공격할지 입력받는 함수이다. State 변수와 재귀함수를 통해 오입력 값을 받지 않고 정확한 입력값을 받을 때까지 작동한다.

Attack

OriginAttack과 MagicAttack 함수를 사용자의 입력에 따라 수행하기 위해 번호를 입력받고 그에 해당하는 case문으로 이동하여 함수를 실행한다.
함수가 공격 입력을 받고 공격까지 수행했다면 Wizard 속성인 actionBool 값을 false 값으로 변경하여 공격을 수행하였음을 표시한다.

compareTo

출력창에서 이름 순서대로 정렬하기 위해 Comparable 인터페이스를 상속받아 Override하였다.

다-ㄴ) GameMaster/ Class

마법사 객체와 몬스터 객체가 여러 개 있어야 하는 상황이다. 때문에 리스트로 객체를 관리하기 위해 wizards, monsters 리스트를 만들었다.
생성자일 때 wizards, monsters 리스트를 생성하고 리스트에 객체를 생성하고 집어넣는데 끝마치면 Collections.sort 함수를 이용해서 이름순으로 정렬한다.

SelectWizard

특정 마법사를 선택하여 공격 명령을 수행하도록 하기 위해서 마법사의 GetActionBool 함수를 통해서 이번 턴에 공격하였는지 검사하고 사용자에게 선택 목록도 나오지 않고 입력하지 못하게 설정하였다.

다-ㄷ) Main / Class

run 변수를 통해서 게임의 진행과 종료를 제어하고 그 안의 상태를 제어하기 위해 state 변수를 통해서 마법사의 공격, 몬스터의 공격, 전투 정산, 마법사의 승리, 몬스터의 승리 이런 흐름으로 이어진다.

2. 코드

```
// 2024_06_12_알고리즘_실습과제
// 한국폴리텍대학_서울정수캠퍼스_인공지능소프트웨어학과
// 2401110252_박지수
// 마법사로 몬스터를 물리치는 게임
import java.util.InputMismatchException;
import java.util.Random;
public class Wizard implements Comparable<Wizard>
{
    private String name = "";
    private int hp;
    private int mp;

    private final int originDMG;
    private final int magicDMG;
    private final int magicCost;
    private final int criticalPer;
    private final int criticalDMG;
    private final int actionList;
```

```
    private int state = 0;
    private int selectMonsterNum = 0;
    private final String[] actions;
    private boolean actionBool = true;
    Random rand = new Random();
    public Wizard()
    {
        name = "Gandalf";
        hp = 80;
        mp = 100;
        originDMG = 5;
        magicDMG = 30;
        magicCost = 30;
        criticalPer = 30;
        criticalDMG = 10;
        actions = new String[] { "1. 기본 공격","2. 마법 공격"};
        actionList = actions.length;
    }
    // 여러개를 생성할 경우 구분할수있게 이름에 숫자를 붙임
    public Wizard(int num)
    {
        this(); // 기본 생성자에 아래 내용을 추가 하겠다는 뜻.

        name += num;
    }
    public int SelectMonster(GameMaster gameMaster)
    {
        // 이 함수의 목적은 어떠한 몬스터를 공격할지 정하고 그 몬스터의 순서가
몇번째인지 반환하는 함수이다.
        // 처음 실행되면 state 변수가 0 부터 시작하고 몬스터들의 정보를 출력한
다음 state 변수를 1 로 바꾼다.
        // 그리고 이 함수를 다시 실행하는 재귀함수이다.
        // try catch 를 통해서 정상적인 숫자값을 받도록 하고 부적격한 값을 얻을
경우 state 변수를 변경하지 않고
        // 다시 재귀하여 정상적인 값을 받아낼 때 까지 반복한다.
        // 정상적인 값을 받아낸다면 state 값을 0 으로 다시 초기화 한 다음 함수를
탈출한다.
        switch(state)
        {
            case 0:
                System.out.println(this.GetName() +"의 차례이다.");

                for (int i = 0; i < gameMaster.monsters.size(); i++) {
                    System.out.printf("%d. 이름: %s  HP: %d\t", i + 1,
gameMaster.monsters.get(i).GetName(),
gameMaster.monsters.get(i).GetHp());
                    System.out.println();
                }
                state = 1;
```

```
                break;
            case 1:
                try
                {
                    do
                    {
                        System.out.printf("공격할 대상을 1 부터 %d 까지만
입력해 주세요\n",gameMaster.monsters.size());
                        selectMonsterNum = gameMaster.sc.nextInt();
                    }
                    while(selectMonsterNum <= 0 || selectMonsterNum >
gameMaster.monsters.size());
                    state = 2;
                }
                catch(InputMismatchException e)
                {
                    gameMaster.sc.nextLine();
                }
                break;
        }
        if (state == 1)
        {
            SelectMonster(gameMaster);
        }
        else
        {
            state = 0;
        }
        return selectMonsterNum;
    }
public void Attack(GameMaster gameMaster,Monster monster)
{
    // 공격 방법을 정하고 그에 맞는 공격 함수를 실행한다. 여기서도 state 변수를
통해 행동을 제어하는데
    // 먼저 안내문이 끝나면 1 로 변화하고 1 에서 어떤 공격을 받을지에 따라 갈린다.
    // 일반공격이면 state 2 로 간 다음 OriginAttack 함수를 실행하고 마법공격이면
state 3 으로 가서 MagicAttack 함수를 실행한다.
    int selectAttack = 0;
    switch(state)
    {
        case 0:
            System.out.println(this.GetName() +"의 공격 방법은 ?");
            for (int i = 0 ; i < this.actions.length ; i++)
            {
                if ( i == 1)
                {
                    System.out.printf("%s(현재
MP%d)",this.actions[i],this.mp);
                }
                else
                {
                    System.out.printf("%s\t", this.actions[i]);
```

```
                }
            }
            System.out.println();
            state = 1;
            break;
        case 1:
            try
            {
                do {
                    System.out.printf("1 부터 %d 까지 입력해
주세요\n",this.actionList);

                selectAttack = gameMaster.sc.nextInt();
                }while(selectAttack <= 0 || selectAttack >
this.actionList );
            }catch(InputMismatchException e)
            {
                gameMaster.sc.nextLine();
            }
            if (selectAttack == 1)
            {
                state = 2;
            }
            else if (selectAttack == 2)
            {
                state = 3;
            }
            break;
        case 2:
            OriginAttack(monster);
            System.out.printf("%s 의 체력이 %d 가 되었다..!",
monster.GetName(), monster.GetHp());
            if (monster.GetHp() <= 0)
            {
                System.out.printf("%s 은(는) 마침내 숨을
거두었다..!\n\n",monster.GetName());
            }
            state = 0;
            break;
        case 3:
            MagicAttack(monster);
            System.out.printf("%s 의 체력이 %d 가 되었다..!",
monster.GetName(), monster.GetHp());
            if (monster.GetHp() <= 0)
            {
                System.out.printf("%s 은(는) 마침내 숨을
거두었다..!\n\n",monster.GetName());
            }
            state = 0;
            break;
    }
    if (state != 0)
    {
```

```
            Attack(gameMaster, monster);
        }

        actionBool = false;
    }
        public void OriginAttack(Monster monster)
        {
            System.out.println("일반공격");
            monster.SetHp(originDMG);
        }

        public void MagicAttack(Monster monster)
        {
            System.out.println("마법공격");
            if (mp >= magicCost)
            {
                SetMp(magicCost);
                if (rand.nextInt(101) < criticalPer)
                {
                    System.out.println("크리티컬 발생 !");
                    monster.SetHp(magicDMG + criticalDMG);
                }
                else
                {
                    monster.SetHp(magicDMG);
                }
            }
            else
            {
                System.out.printf("가지고 있는 MP 에 비해 요구량이 큰 공격을
사용하여 %s 을(를) 사용합니다.\n",this.actions[0]);
                OriginAttack(monster);
            }
        }
        public int GetHp()
        {
            return hp;
        }

        public void SetHp(int value)
        {
            hp -= value;
        }
        public void SetMp(int value)
        {
            mp -= value;
        }
        public int GetMp()
        {
            return mp;
        }
        public String GetName()
        {
            return name;
```

```
    }
    public int GetActionList()
    {
        return actionList;
    }
    public void SetActionFalse()
    {
        actionBool = false;
    }
    public void SetActionTrue()
    {
        actionBool = true;
    }
    public boolean GetActionBool()
    {
        return actionBool;
    }


    // 마법사의 순서를 이름순으로 오름차순 정렬하기 위해 Comparable 인터페이스를 상속받았다.
    @Override
    public int compareTo(Wizard wiz)
    {
        return this.name.compareTo(wiz.name);
    }
}
```

Wizard 클래스

```
public class Monster implements Comparable<Monster>
{
    private String name = "";
    private int hp;
    private final int originDMG;

    public Monster(String name)
    {
        this.name = name;
        hp = 100;
        originDMG = 10;
    }

    public int GetDMG()
    {
        return originDMG;
    }

    public int GetHp()
    {
        return hp;
    }

    public void SetHp(int value)
    {
        hp -= value;
```

```
    }

    public void Attack(Wizard wizard)
    {
        wizard.SetHp(originDMG);
        System.out.println(name +"이(가)"+wizard.GetName()+"을(를)
공격하여"+ " HP 가" + wizard.GetHp()+" 되었습니다.");
        if (wizard.GetHp() <= 0)
        {
            System.out.println(wizard.GetName()+"의 목숨이 끊어졌다...!");
        }
    }

    public String GetName()
    {
        return name;
    }

    // 몬스터의 순서를 이름순으로 오름차순 정렬하기 위해 Comparable 인터페이스를
상속받았다.
    @Override
    public int compareTo(Monster mon) {
        return this.name.compareTo(mon.name);
    }

}
```

Monster 클래스

```
import java.util.*;

public class GameMaster
{
    Scanner sc;
    List<Monster> monsters;
    List<Wizard> wizards;
    int monsterCount = 0;
    int wizardCount = 0;
    int actionWizardCount = 0;
    int setUnitCountState = 0;
    private final int maxWizardCount = 9;
    private final int maxMonsterCount = 9;
    public GameMaster()
    {
        sc = new Scanner(System.in);
        monsters = new ArrayList<>();
        wizards = new ArrayList<>();
        SetUnitCount();
        for(int i = 0, j = 0, k = 0 ; i < monsterCount; i++) {
            if (i % 2 == 0) {
                k++;
                monsters.add(new Monster("Orc" + "%d".formatted(k)));
            } else {
                j++;
```

```
                monsters.add(new Monster("Goblin" + "%d".formatted(j)));
            }
        }
        Collections.sort(monsters);
        for(int i = 0; i < wizardCount; i++)
        {
            wizards.add(new Wizard(i+1));
        }
        Collections.sort(wizards);
    }
    public void RemoveMonster(List<Monster> monsters)
    {
        for (int i = monsters.size()-1 ; i >= 0; i--)
        {
            if (monsters.get(i).GetHp() <= 0) {
                monsters.remove(i);
                monsterCount--;
            }
        }
    }
    public void RemoveWizard(List<Wizard> wizards)
    {
        for (int i = wizards.size()-1 ; i >= 0; i--)
        {
            if (wizards.get(i).GetHp() <= 0) {
                wizards.remove(i);
                wizardCount--;
            }
        }
    }
public void SetActionTrue()
    {
        for (Wizard wiz : wizards)
        {
            wiz.SetActionTrue();
        }
    }

    public void SetUnitCount() {

        switch (setUnitCountState) {
            case 0:
                try {
                    do {
                        System.out.println("마법사의 수와 몬스터의 수를 입력해
주세요.");

                        System.out.printf("마법사의 수(최대 %d) : ",
maxMonsterCount);
                        this.wizardCount = sc.nextInt();
                    }
                    while (wizardCount <= 0 || wizardCount >
maxWizardCount);
                    setUnitCountState = 1;
                } catch (InputMismatchException e) {
                    sc.nextLine();
```

```
                }
                SetUnitCount();
                break;
            case 1:
                try {
                    do {
                        System.out.println("몬스터의 수를 입력해 주세요.");
                        System.out.printf("몬스터의 수(최대 %d) : ",
maxMonsterCount);
                        this.monsterCount = sc.nextInt();
                    }
                    while (monsterCount <= 0 || monsterCount >
maxMonsterCount);
                    setUnitCountState = 2;
                } catch (InputMismatchException e) {
                    sc.nextLine();
                }

                SetUnitCount();
                break;
            case 2:
                System.out.printf("마법사 %d 개 몬스터 %d 개
생성하였습니다.\n", wizardCount, monsterCount);
        }

    }
    public int SelectWizard()
    {
        // 사용한 객체들은 삭제하고 다시 추가하여 제일 뒤로 보내고 얼마나 보냈는지
기록한다.
        // 반복문 횟수는 총 리스트의 크기에 사용한 갯수만큼 차감해 반복문을 적게
실행하게 만들어
        // 뒤로 간 객체들은 출력되지 않게 만든다.
        int selectWizard = 0;
        Wizard tempWizard = null;
          {

            do {
                try {
                        System.out.println("사용할 수 있는 마법사입니다.");
                        for (int i = 0; i < wizards.size() -
actionWizardCount; i++)
                        {
                            if (!wizards.get(i).GetActionBool())
                            {
                                tempWizard = wizards.get(i);
                                wizards.remove(tempWizard);
                                wizards.add(tempWizard);
                                actionWizardCount++;
                            }
                        }
                        for (int i = 0; i < wizards.size() -
```

```
actionWizardCount; i++)
                    {

System.out.printf("%d\t 이름: %s\tHP: %d\tMP: %d\n"
                                , i + 1, wizards.get(i).GetName(),
wizards.get(i).GetHp(), wizards.get(i).GetMp());
                    }
                    System.out.printf("1 부터 %d 까지 입력해 주세요\n",
wizards.size() - actionWizardCount);
                    selectWizard = sc.nextInt();
                    }
                catch (Exception e) {
                    selectWizard = 0;
                    sc.nextLine();
                }
            }while (selectWizard <= 0 || selectWizard > wizards.size()
- actionWizardCount);
        }
        return selectWizard-1;
    }
    public void InsertWizard(int number)
    {
        for(int i = 0; i < number; i++)
        {
            wizards.add(new Wizard(i+wizardCount));
        }
        wizardCount += number;
    }
    public void InsertMonster(int number)
    {
        for(int i = 0, j = 0, k = 0 ; i < number; i++)
        {
            if ( i % 2 == 0)
            {
                k++;
                monsters.add(new
Monster("Orc"+"%d".formatted(k+monsterCount)));
            }
            else
            {
                j++;
                monsters.add(new
Monster("Goblin"+"%d".formatted(j+monsterCount)));
            }
        }
        monsterCount += number;
        Collections.sort(this.monsters);
    }
    public void ActionWizardCountClear()
    {
        actionWizardCount = 0;
    }
}
```

GameMaster 클래스

```
import java.util.Random;
public class Main
{

    public static void main(String[] args)
    {
        int state = 0;
        int turn = 1;
        boolean run = true;
        Random rand = new Random();
        GameMaster gameMaster = new GameMaster();

        while(run)
        {
            switch (state)
            {
                // 마법사가 때리기
                case 0:
                    for (int i = 0; i < gameMaster.wizardCount; i++)
                    {
                        int indexWizard = gameMaster.SelectWizard();
                        int selectMon =
gameMaster.wizards.get(indexWizard).SelectMonster(gameMaster);

gameMaster.wizards.get(indexWizard).Attack(gameMaster,gameMaster.mons
ters.get(selectMon-1));
                        gameMaster.RemoveMonster(gameMaster.monsters);
                        if (gameMaster.monsters.isEmpty())
                        {
                            System.out.println("적을 전부 죽였다.");
                            break;
                        }
                    }
                    if (gameMaster.monsters.isEmpty())
                    {
                        state = 3;
                        break;
                    }
                    else
                    {
                        state = 1;
                    }
                    break;
                    // 몬스터가 때리기
                case 1:
                    gameMaster.SetActionTrue();
                    gameMaster.ActionWizardCountClear();
                    for (int i = 0 ; i< gameMaster.monsterCount; i++)
                    {
                        int randWizard =
rand.nextInt(gameMaster.wizardCount);

gameMaster.monsters.get(i).Attack(gameMaster.wizards.get(randWizard))
;
                        gameMaster.RemoveWizard(gameMaster.wizards);
```

```
                    if (gameMaster.wizards.isEmpty())
                    {
                        System.out.println("플레이어 유닛이 전부
죽었다.\n");
                        break;
                    }
                }
                if (gameMaster.wizards.isEmpty())
                {
                    state = 4;
                    break;
                }
                state = 2;
                break;
            case 2:
                System.out.printf("%d 번째 전투가 끝났다.\n",turn);
                System.out.println("각 팀의 상태");
                for (int i = 0; i < gameMaster.wizardCount; i++)
                {
                    System.out.printf("이름: %s\tHP: %d\tMP: %d\n",
                            gameMaster.wizards.get(i).GetName(),
                            gameMaster.wizards.get(i).GetHp(),
                            gameMaster.wizards.get(i).GetMp());
                }
                System.out.println();
                for (int i = 0; i < gameMaster.monsterCount; i++)
                {
                    System.out.printf("이름: %s\tHP: %d\n",
                            gameMaster.monsters.get(i).GetName(),
                            gameMaster.monsters.get(i).GetHp());
                }
                System.out.println("\n\n");
                turn++;
                state = 0;
                break;
            case 3:
                System.out.printf("%d 번째 전투에서 마법사의 승리로 끝이
났다.",turn);
                run = false;
                break;
            case 4:
                System.out.printf("%d 번째 전투에서 몬스터의 승리로 끝이
났다.",turn);
                run = false;
                break;
            }
        }
        if (state == 3)
        {
            System.out.println("승리");
        }
```

```
        else if (state == 4)
        {
            System.out.println("패배");
        }
    }
}
```

Main 클래스

3. 결과 스크린샷

```
마법사의 수와 몬스터의 수를 입력해 주세요.
마법사의 수(최대 9) : 2
몬스터의 수를 입력해 주세요.
몬스터의 수(최대 9) : 2
마법사 2개 몬스터 2개 생성하였습니다.
사용할 수 있는 마법사입니다.
1   이름: Gandalf1   HP: 80  MP: 100
2   이름: Gandalf2   HP: 80  MP: 100
1 부터 2 까지 입력해 주세요
1
Gandalf1의 차례이다.
1. 이름: Goblin1  HP: 100
2. 이름: Orc1  HP: 100
공격할 대상을 1 부터 2 까지만 입력해 주세요
2
Gandalf1의 공격 방법은 ?
1. 기본 공격  2. 마법 공격(현재 MP100)
1 부터 2 까지 입력해 주세요
2
마법공격
Orc1의 체력이 70가 되었다..!
사용할 수 있는 마법사입니다.
1   이름: Gandalf2   HP: 80  MP: 100
1 부터 1 까지 입력해 주세요
1
Gandalf2의 차례이다.
1. 이름: Goblin1  HP: 100
2. 이름: Orc1  HP: 70
공격할 대상을 1 부터 2 까지만 입력해 주세요
2
Gandalf2의 공격 방법은 ?
1. 기본 공격  2. 마법 공격(현재 MP100)
1 부터 2 까지 입력해 주세요
2
마법공격
Orc1의 체력이 40가 되었다..!
Goblin1이(가)Gandalf1을(를) 공격하여 HP가70 되었습니다.
Orc1이(가)Gandalf2을(를) 공격하여 HP가70 되었습니다.
1번째 전투가 끝났다.
각 팀의 상태
이름: Gandalf2   HP: 70  MP: 70
이름: Gandalf1   HP: 70  MP: 70

이름: Goblin1   HP: 100
이름: Orc1   HP: 40


사용할 수 있는 마법사입니다.
1   이름: Gandalf2   HP: 70  MP: 70
2   이름: Gandalf1   HP: 70  MP: 70
1 부터 2 까지 입력해 주세요
```

처음 작동 방식 및 공격 방식

```
Gandalf1의 차례이다.
1. 이름: Orc1  HP: 100
공격할 대상을 1 부터 1 까지만 입력해 주세요
1
Gandalf1의 공격 방법은 ?
1. 기본 공격  2. 마법 공격(현재 MP10)
1 부터 2 까지 입력해 주세요
2
마법공격
가지고 있는 MP에 비해 요구량이 큰 공격을 사용하여 1. 기본 공격을(를) 사용합니다.
일반공격
Orc1의 체력이 95가 되었다..!
```

MP가 부족한 상황에서 마법공격을 사용한 경우

```
마법사의 수와 몬스터의 수를 입력해 주세요.
마법사의 수(최대 9) : ␣
마법사의 수와 몬스터의 수를 입력해 주세요.
마법사의 수(최대 9) : 10
마법사의 수와 몬스터의 수를 입력해 주세요.
마법사의 수(최대 9) : 1
몬스터의 수를 입력해 주세요.
몬스터의 수(최대 9) : ␣
몬스터의 수를 입력해 주세요.
몬스터의 수(최대 9) : 10
몬스터의 수를 입력해 주세요.
몬스터의 수(최대 9) : 1
마법사 1개 몬스터 1개 생성하였습니다.
사용할 수 있는 마법사입니다.
1   이름: Gandalf1   HP: 80  MP: 100
1 부터 1 까지 입력해 주세요
␣
사용할 수 있는 마법사입니다.
1   이름: Gandalf1   HP: 80  MP: 100
1 부터 1 까지 입력해 주세요
2
사용할 수 있는 마법사입니다.
1   이름: Gandalf1   HP: 80  MP: 100
1 부터 1 까지 입력해 주세요
1
Gandalf1의 차례이다.
1. 이름: Orc1  HP: 100
공격할 대상을 1 부터 1 까지만 입력해 주세요
␣
공격할 대상을 1 부터 1 까지만 입력해 주세요
2
공격할 대상을 1 부터 1 까지만 입력해 주세요
1
Gandalf1의 공격 방법은 ?
1. 기본 공격  2. 마법 공격(현재 MP100)
1 부터 2 까지 입력해 주세요
```

```
1. 기본 공격  2. 마법 공격(현재 MP100)
1 부터 2 까지 입력해 주세요
␣
1 부터 2 까지 입력해 주세요
3
1 부터 2 까지 입력해 주세요
2
마법공격
크리티컬 발생 !
Orc1의 체력이 60가 되었다..!
Orc1이(가)Gandalf1을(를) 공격하여 HP가70 되었습니다.
1번째 전투가 끝났다.
각 팀의 상태
이름: Gandalf1   HP: 70  MP: 70

이름: Orc1   HP: 60



사용할 수 있는 마법사입니다.
1   이름: Gandalf1   HP: 70  MP: 70
1 부터 1 까지 입력해 주세요
```

잘못된 입력을 했을 경우

Orc1이(가)Gandalf1을(를) 공격하여 HP가0 되었습니다. Gandalf1의 목숨이 끊어졌다...! 플레이어 유닛이 전부 죽었다. 8번째 전투에서 몬스터의 승리로 끝이 났다.패배	Orc1의 체력이 -10가 되었다..! Orc1은(는) 마침내 숨을 거두었다..! 적을 전부 죽였다. 1번째 전투에서 마법사의 승리로 끝이 났다.승리

전투가 끝났을 경우

4. 고찰

저번 과제와 같은 주제를 가지고 코드를 새로 작성하면 어떨지 궁금함을 가지고 과제에 임했었다. 지난 과제에서는 클래스 활용을 위해 상속과 인터페이스를 사용하여 코드를 제출하였지만, 이번 과제에서는 여러 객체를 관리하여 많은 마법사와 몬스터를 상대해 보는 전투방식과 지난 과제의 피드백이었던 상태 변수를 사용하여 제어하라는 것이 기억에 남아 이를 생각하고 과제를 진행하였다. 똑같은 주제였지만 총 5개 정도의 언덕과 영감이 있었다.

첫번째로는 클래스 속성을 private 설정하고 함수를 이용하여 값을 읽어오는 것이 낯설기도 하고 수고스러움이 강했다. 새 함수를 작성해야 하고, 함수를 통해 접근해야 했기 때문이다. 그러나 코드의 양과 복잡성이 늘어남에 따라 직접적으로 속성을 접근하여 변경하는 것보다는 함수를 통해 접근하는 것이 유지관리 측면에서 효과적이라는 것을 과제를 통해 깨달았다. 이 방식을 사용하여 코드를 작성하고 수정하다 보니, 버그가 발생했을 때 함수나 변수가 사용, 변경되는 것을 쉽게 추척할 수 있었고, 오류가 발생한 문제 블록을 특정할 수 있었다.

두번째로는 몬스터 객체를 생성하고 이름도 부여하다 보니 너무 난잡하게 보여 이름순으로 정렬하고 싶어 정렬 방법도 찾아보았다. Monster라는 사용자 지정 클래스의 List를 정렬하기 위해서는 Comparable 인터페이스를 상속받고 compareTo 함수를 재정의 한 다음 Collection.sort 함수를 통해서 속성들을 비교할 수 있었다.

세번째로는 Try Catch문으로 명령을 정수값만 입력받기 위해서 사용하고 에러가 났을 때 다시 함수를 실행하기 위해서 재귀함수를 사용했었지만, 재귀함수 진입점 위치와 종료 시점을 잘못 지정하였다.
정수이지만 입력 받은 정수의 크기가 요구치 보다 작거나 큰 값을 입력한 상황이 발생한 다음 정수가 아닌 값을 입력할 때 재귀를 하게 되었는데 이때 정상적인 입력값을 입력하면 스택프레임에 쌓인 잘못 입력한 값들이 다 우수수 나와서 스택의 자료구조 순서로 리턴되는 상황이 나왔다. 오입력된 값들은 버려지도록 while문 범위를 다시 정렬하고 재귀를 없애 정상적으로 작동하게 하였다. 재귀함수를 사용할 때는 진입 위치로 돌아왔을 때 어떤 값을 가지고 진행될지 고민하여 작성해야겠다.

네번째로는 Wizard 클래스 속성에 공격 여부를 기록하기 위한 actionBool 변수를 생성하였는데 이 속성을 이용하면 공격한 마법사들의 리스트를 새로 생성하고 그 객체를 공격한 리스트에 추가할 수 있다고 생각했다. 거기다 기존 공격하지 않은 마법사들 리스트에서는 삭제한다면 깔끔하게 사용할 수 있는 마법사만 출력할 수 있다고 생각했고 실제로 작성도 했다.

하지만 기존의 for each을 사용하여 리스트 원소를 삭제를 하니 에러가 발생하였고 해결법은 Iterator 인터페이스를 통해서 제거를 하니 안전하게 삭제가 되었다. 삭제도 되고 출력도 잘 되었지만, 코드를 리뷰하는 도중 이러면 굳이 새로운 리스트를 만들지 않아도 그 속성만 검사한다면 리소스 낭비를 하지 않고 똑같이 구현할 수 있었다는 것을 깨달았다. 문제를 해결하기 위해 빙빙 돌아간 격이었다. 다시 한번 기획의 중요성과 왜 이렇게 해야 하는지 되물어 보는 습관을 지닐 필요성을 느꼈다.

마지막으로 공격방식, state 변수를 통한 제어를 이번 게임에서 많이 사용하게 되었는데 10일 C언어 수업 시간 중 enum이라는 문법을 습득하였다. 수업 시간에는 간단히 넘어갔지만 이를 보는 순간 과제에 적합함이 떠올랐고 예제도 찾아보니 코드가 길어질수록 가독성이 좋아지는 효과를 확인했다. 다음 과제에서는 enum을 사용하여 명시적으로 상태를 시인해 좋은 코드를 작성할 수 있을 것 같다.