객체지향프로그래밍 HW#1

추민솔, 최지헌 TA
oop-ta_grp@ajou.ac.kr

2025-04-16

목차

1 과제

1.1 목표

- 객체지향 프로그래밍 개념인 상속(Inheritance), 다형성(Polymorphism), 캡슐화(Encapsulation), 추상화(Abstraction)를 활용하여 30층 던전 탐험 게임을 구현한다.
- 클래스 간의 관계와 상호작용을 이해하고, 스켈레톤 코드를 바탕으로 게임의 핵심 기능을 완성한다.

1.2 게임 개요

- 플레이어는 1층부터 시작하여 30층까지 던전을 탐험하며, 각 층은 5개의 방으로 구성되어 있다.
- 몬스터와의 전투를 통해 경험치를 얻고 레벨업하며 아이템을 획득할 수 있다.
- 10, 20, 30층마다 특별한 보스 몬스터가 등장하며, 30층 보스를 물리치면 게임이 클리어된다.
- 플레이어는 체력, 공격력, 방어력 등의 능력치를 가지며, 무기와 방어구를 장착할 수 있다.

1.3 클래스 구조

전체 클래스 구조는 아래 다이어그램과 같다.

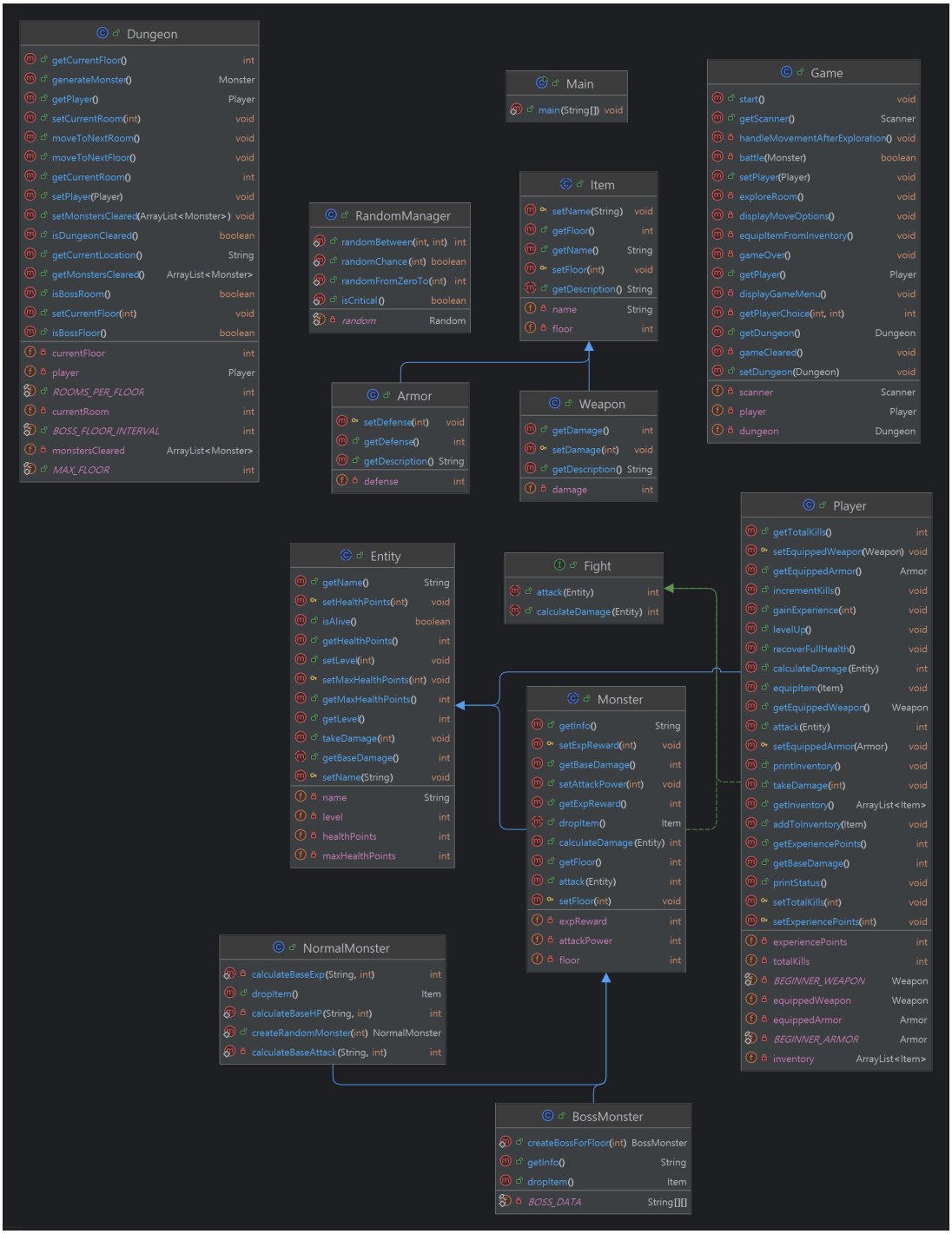

Figure 1: 던전 게임 클래스 다이어그램

1.4 제공되는 클래스

아래의 클래스들은 완성된 형태로 제공된다.

1.4.1 Main.java

- 게임의 진입점으로, 플레이어 이름을 입력받고 Game 객체를 생성하여 게임을 시작한다.

1.4.2 Fight.java (인터페이스)

- Entity 간의 전투에 필요한 메소드를 정의한 인터페이스이다.
- attack(Entity target) : 대상에게 공격을 가하는 메소드
- calculateDamage(Entity target) : 레벨 차이, 장비 등을 고려하여 최종 데미지를 계산하는 메소드

1.4.3 RandomManager.java

- 게임 내 확률적 요소와 랜덤값을 처리하는 유틸리티 클래스이다.
- 다음의 정적 메소드를 제공한다:
  - boolean isCritical() : 30% 확률로 치명타(critical hit) 여부를 반환
  - int randomFromZeroTo(int max) : 0부터 max까지의 랜덤 정수를 반환
  - int randomBetween(int min, int max) : min부터 max까지의 랜덤 정수를 반환
  - boolean randomChance(int probability) : 지정된 확률(0-100)로 성공 여부를 반환

1.4.4 Game.java

- 게임의 핵심 로직과 사용자 인터페이스를 관리하는 클래스이다.
- 플레이어와 던전 객체를 생성하고, 사용자 입력에 따라 게임 진행을 제어한다.
- 메뉴 표시, 탐험, 전투, 아이템 장착 등 게임의 주요 기능을 구현한다.
- 게임 오버나 게임 클리어와 같은 특수 상황을 처리한다.

필드

- private Player player : 게임 플레이어
- private Dungeon dungeon : 게임 던전
- private Scanner scanner : 사용자 입력 처리

메소드

- void start() : 게임을 시작하고 주요 게임 루프를 실행하는 메소드
- void displayGameMenu() : 게임 메뉴를 표시하는 메소드
- void exploreRoom() : 현재 방을 탐험하는 메소드
- void handleMovementAfterExploration() : 탐험 후 이동 처리를 담당하는 메소드
- void displayMoveOptions() : 이동 옵션을 표시하는 메소드
- boolean battle(Monster monster) : 몬스터와의 전투를 처리하는 메소드

- void equipItemFromInventory() : 인벤토리에서 아이템을 장착하는 메소드
- void gameOver() : 게임 오버 처리를 담당하는 메소드
- void gameCleared() : 게임 클리어 처리를 담당하는 메소드
- int getPlayerChoice(int min, int max) : 플레이어의 선택을 받아 처리하는 메소드

1.5 구현해야 할 클래스

아래 클래스들은 스켈레톤 코드가 제공된다.

1.5.1 Entity.java (추상 클래스)

- 게임 내 모든 생명체(플레이어와 몬스터)의 공통 특성을 정의하는 추상 클래스이다.
- 이름, 체력, 레벨과 같은 기본 속성을 관리하며, 데미지를 받거나 생존 여부를 확인하는 기본 기능을 제공한다.
- 각 하위 클래스(Player, Monster)는 고유한 방식으로 기본 공격력을 계산해야 한다.

필드

- private String name : 엔티티(플레이어, 몬스터)의 이름
- private int healthPoints : 현재 체력
- private int maxHealthPoints : 최대 체력
- private int level : 레벨

구현해야 할 메소드

- void takeDamage(int damage) : 데미지를 받아 체력을 감소시키는 메소드
- boolean isAlive() : 현재 생존 여부(체력이 0 초과인지)를 반환하는 메소드
- abstract int getBaseDamage() : 기본 공격력을 반환하는 추상 메소드

1.5.2 Item.java (추상 클래스)

- 게임 내 모든 아이템의 공통 특성을 정의하는 추상 클래스이다.
- 이름과 획득한 층 정보를 관리하며, 각 하위 클래스(Weapon, Armor)는 자신만의 방식으로 아이템 설명을 제공해야 한다.
- 아이템은 인벤토리에 저장되거나 장착되어 플레이어의 능력치에 영향을 줄 수 있다.

필드

- private String name : 아이템 이름
- private int floor : 아이템을 획득한 층

구현해야 할 메소드

- abstract String getDescription() : 아이템 설명을 반환하는 추상 메소드

1.5.3 Weapon.java

- 플레이어가 장착할 수 있는 무기를 나타내는 클래스이다.
- Item 클래스를 상속받으며, 추가로 공격력 속성을 가진다.
- 무기는 플레이어의 공격력을 추가로 증가시킨다.

필드

- private int damage : 무기의 공격력

구현해야 할 메소드

- String getDescription() : 무기 설명을 반환

1.5.4 Armor.java

- 플레이어가 장착할 수 있는 방어구를 나타내는 클래스이다.
- Item 클래스를 상속받으며, 추가로 방어력 속성을 가진다.
- 방어구는 플레이어가 받는 데미지를 감소시킨다.

필드

- private int defense : 방어구의 방어력

구현해야 할 메소드

- String getDescription() : 방어구 설명을 반환

1.5.5 Monster.java (추상 클래스)

- 게임 내 모든 몬스터의 공통 특성을 정의하는 추상 클래스이다.
- Entity 클래스를 상속받고 Fight 인터페이스를 구현하여 전투 기능을 제공한다.
- 몬스터는 공격력, 경험치 보상, 출현 층 등의 추가 속성을 가지며, 보스 몬스터 처치 시 아이템을 드롭할 수 있다.
- 하위 클래스(NormalMonster, BossMonster)는 각각 다른 방식으로 생성되고 보스 몬스터는 아이템을 드롭한다.

필드

- private int attackPower : 몬스터의 공격력
- private int expReward : 몬스터 처치 시 얻는 경험치
- private int floor : 몬스터가 등장하는 층

구현해야 할 메소드

- int attack(Entity target) : 대상에게 공격을 가하는 메소드
- int calculateDamage(Entity target) : 레벨 차이 등을 고려하여 데미지를 계산하는 메소드
- String getInfo() : 몬스터의 정보(이름, 레벨, 체력)를 문자열로 반환하는 메소드
- abstract Item dropItem() : 몬스터가 드롭하는 아이템을 반환하는 추상 메소드

1.5.6 NormalMonster.java

- 일반적인 몬스터를 나타내는 클래스로, Monster 클래스를 상속받는다.
- 층수에 따라 다양한 종류(슬라임, 고블린, 오크, 해골, 좀비)의 몬스터가 무작위로 생성된다.
- 몬스터의 종류와 레벨에 따라 체력, 공격력, 경험치가 다르게 설정된다.
- 일반 몬스터는 아이템을 드롭하지 않는다.

필드

- 추가 필드 없음 (Monster 클래스의 필드 상속)

구현해야 할 메소드

- static NormalMonster createRandomMonster(int floor) : 해당 층에 맞는 일반 몬스터를 무작위로 생성하는 정적 메소드
- static int calculateBaseHP(String name, int level) : 몬스터 이름과 레벨에 따른 기본 체력을 계산하는 정적 메소드
- static int calculateBaseAttack(String name, int level) : 몬스터 이름과 레벨에 따른 기본 공격력을 계산하는 정적 메소드
- static int calculateBaseExp(String name, int level) : 몬스터 이름과 레벨에 따른 기본 경험치를 계산하는 정적 메소드
- Item dropItem() : 일반 몬스터는 아이템을 드롭하지 않으므로 null을 반환

1.5.7 BossMonster.java

- 보스 몬스터를 나타내는 클래스로, Monster 클래스를 상속받는다.
- 10층, 20층, 30층과 같이 특정 층의 마지막 방에 등장하는 강력한 몬스터이다.
- 일반 몬스터보다 더 높은 체력, 공격력, 경험치를 가지고 있다.
- 보스 몬스터를 처치하면 특별한 무기나 방어구를 획득할 수 있다.

필드

- private static final String[][] BOSS_DATA : 각 보스의 정보(이름, 체력, 공격력, 경험치)를 담은 2차원 배열

구현해야 할 메소드

- static BossMonster createBossForFloor(int floor) : 해당 층에 맞는 보스 몬스터를 생성하는 정적 메소드
- Item dropItem() : 보스 몬스터가 드롭하는 아이템(무기 또는 방어구)을 반환하는 메소드
- String getInfo() : 부모 클래스의 getInfo() 결과를 반환하는 메소드

1.5.8 Player.java

- 게임의 주인공인 플레이어를 나타내는 클래스이다.
- Entity 클래스를 상속받고 Fight 인터페이스를 구현하여 전투 기능을 제공한다.
- 플레이어는 경험치를 모아 레벨업하고, 아이템을 획득하고 장착하여 능력치를 향상시킬 수 있다.

필드

- private int experiencePoints : 플레이어의 현재 경험치
- private ArrayList<Item> inventory : 플레이어의 인벤토리
- private Weapon equippedWeapon : 장착 중인 무기
- private Armor equippedArmor : 장착 중인 방어구
- private int totalKills : 처치한 총 몬스터 수

구현해야 할 메소드

- void gainExperience(int exp) : 경험치를 획득하고 필요시 레벨업하는 메소드
- void levelUp() : 레벨을 올리고 능력치를 상승시키는 메소드
- void equipItem(Item item) : 아이템을 장착하는 메소드
- int attack(Entity target) : 대상에게 공격을 가하는 메소드

- int calculateDamage(Entity target) : 레벨 차이, 무기 등을 고려하여 데미지를 계산하는 메소드
- void takeDamage(int damage) : 방어구를 고려하여 데미지를 감소시키는 메소드
- void addToInventory(Item item) : 인벤토리에 아이템을 추가하는 메소드
- void incrementKills() : 처치한 몬스터 수를 증가시키는 메소드
- void recoverFullHealth() : 체력을 완전히 회복시키는 메소드
- void printStatus() : 플레이어 상태를 출력하는 메소드
- void printInventory() : 인벤토리를 출력하는 메소드

1.5.9 Dungeon.java

- 게임의 던전 환경을 관리하는 클래스이다.
- 던전의 구조(층, 방), 플레이어의 현재 위치, 몬스터 생성 등을 담당한다.
- 플레이어가 층과 방 사이를 이동할 수 있게 하며, 특정 층마다 보스 몬스터가 등장하도록 관리한다.
- 30층의 보스를 물리치면 던전이 클리어된 것으로 간주한다.

필드

- public static final int MAX_FLOOR : 던전의 최대 층수 (30)
- public static final int ROOMS_PER_FLOOR : 각 층의 방 개수 (5)
- public static final int BOSS_FLOOR_INTERVAL : 보스가 등장하는 층의 간격 (10)
- private int currentFloor : 현재 층
- private int currentRoom : 현재 방
- private Player player : 던전을 탐험 중인 플레이어
- private ArrayList<Monster> monstersCleared : 처치한 몬스터 목록

구현해야 할 메소드

- Monster generateMonster() : 현재 위치에 맞는 몬스터를 생성하는 메소드
- void moveToNextRoom() : 다음 방으로 이동하는 메소드
- void moveToNextFloor() : 다음 층으로 이동하는 메소드
- boolean isDungeonCleared() : 던전 클리어 여부를 확인하는 메소드
- boolean isBossFloor() : 현재 층이 보스 층인지 확인하는 메소드
- boolean isBossRoom() : 현재 방이 보스 방인지 확인하는 메소드
- String getCurrentLocation() : 현재 위치를 문자열로 반환하는 메소드

1.6 구현 요구사항

- Entity, Item과 같은 추상 클래스는 추상 메소드를 포함해야 한다.
- Fight 인터페이스는 전투 관련 메소드를 정의하며, Player와 Monster 클래스에서 이를 구현해야 한다.
- 레벨업 시스템은 경험치가 100 이상이 되면 레벨이 1 증가하고 경험치를 100만큼 감소시킨다.(경험치가 99이하가 될 때까지 이 과정을 반복한다.)
- 전투 시스템은 레벨 차이에 따른 데미지 보정, 치명타, 방어력 등을 고려해야 한다.
- 몬스터는 층과 종류에 따라 다른 능력치(체력, 공격력, 경험치)를 가진다.
- 보스 몬스터는 10, 20, 30층에 등장하며, 특별 아이템을 드롭한다.

1.7 게임 플레이 예시

```
30층 던전 탐험 게임에 오신 것을 환영합니다!
플레이어 이름을 입력하세요:
용사

===== 30층 던전 탐험 게임을 시작합니다 =====
용사님, 던전 탐험을 시작합니다.
10층, 20층, 30층마다 보스가 있고, 30층 보스를 물리치면 게임 클리어입니다.

===== 현재 위치: 1층 1번 방 =====
1. 탐험하기
2. 상태 확인
3. 인벤토리 확인
4. 게임 종료
선택: 1

===== 1층 1번 방 탐험 =====
슬라임 (레벨 0) - HP: 15/15가 나타났습니다!

===== 전투 준비 =====
1. 전투를 한다.
2. 도망간다. (30% 확률로 성공)
선택: 1

===== 전투 시작 =====
용사의 턴입니다.
치명타!
용사가 16의 데미지를 입혔습니다.
```

```
슬라임가 16의 데미지를 받았습니다. 남은 체력: 0
슬라임 (레벨 0) - HP: 0/15를 처치했습니다!
경험치 7를 획득했습니다.
용사가 7 경험치를 획득했습니다.

다음 행동을 선택하세요:
1. 다음 방으로 이동
2. 상태 확인
3. 인벤토리 확인
선택:
```

1.8 제출 요구사항

- 모든 Java 소스 파일(.java)과 보고서 파일을 "이름_학번"을 파일명으로 한 하나의 zip 파일로 압축하여 제출한다.
- 스켈레톤 코드 이외에 추가적인 클래스나 메소드를 구현한 경우, 그 이유와 기능을 설명하여 보고서에 추가한다.

1.9 보고서 가이드

- 파일 포맷은 word 혹은 HWP 으로 작성하여 제출
- 파일명은 HW1_보고서_본인학번 형식으로 제출
- 보고서 목차
  1. **개요**
     - 간단한 과제 설명
  2. **문제 정의**
     - 요구사항 설명
  3. **구현**
     - 각 클래스에 대한 설명 (멤버들에 대한 설명 포함) 과, 요구조건들을 달성하기 위해 이 클래스 및 멤버를 사용한 내용 (즉, 요구조건을 달성하기 위해 클래스/클래스 멤버를 어떻게 사용하였는지를 설명)
  4. **결과**
     - 테스트 방법 및 최종 결과 (예: screenshot 등)
  5. **결론**
     - 과제를 통해 배운점 등