**2022-1**

**객체지향 프로그래밍**

**실습**

**Homework 2  
(Week 7)**

Interface

**이 문서의 버전은 6입니다.**

**(버전 5 생성 시각: 2022년 4월 14일 18시 56분)**

**Composed by:**

**WISE Research Lab Ajou University**

Contents

[1.1. 실습 목표 4](#_Toc100824817)

[1.2. 인터페이스 (interface) 4](#_Toc100824818)

[1.2.1. 인터페이스란? 4](#_Toc100824819)

[1.2.2. 자바의 인터페이스 4](#_Toc100824820)

[1.2.3. 인터페이스와 다형성 4](#_Toc100824821)

[1.2.4. default 인터페이스 메서드 6](#_Toc100824822)

[II. 실습 과제 8](#_Toc100824823)

[2.1. 과제 제출 방법 (요약) 8](#_Toc100824824)

[2.2. 프로그램 구현 8](#_Toc100824825)

[가) 배경 8](#_Toc100824826)

[나) 모델링 9](#_Toc100824827)

[ Entity 클래스 10](#_Toc100824828)

[ Champion 클래스 10](#_Toc100824829)

[ Dealer 클래스 11](#_Toc100824830)

[ Supporter 클래스 12](#_Toc100824831)

[ TreasureBox 클래스 13](#_Toc100824832)

[ MovingTresaureBox 클래스 14](#_Toc100824833)

[ ThickTreasureBox 클래스 14](#_Toc100824834)

[ LifeProperty 클래스 14](#_Toc100824835)

[ LocationProperty 클래스 15](#_Toc100824836)

[ BaseSkill 클래스 15](#_Toc100824837)

[ DealerSkill 클래스 16](#_Toc100824838)

[ SupporterSkill 클래스 17](#_Toc100824839)

[ DoubledDefenseSkill 클래스 18](#_Toc100824840)

[ Attackable 인터페이스 18](#_Toc100824841)

[ Hittable 인터페이스 18](#_Toc100824842)

[ Movable 인터페이스 18](#_Toc100824843)

[ SkillActivatable 인터페이스 19](#_Toc100824844)

[다) 공격 처리 19](#_Toc100824845)

[라) 프로그램의 입력 21](#_Toc100824846)

[마) 프로그램 출력 22](#_Toc100824847)

[바) 특수 처리 23](#_Toc100824848)

[2.3. 프로그램 실행 예시 24](#_Toc100824849)

[2.3.1. 시나리오 1 24](#_Toc100824850)

[ 입력 24](#_Toc100824851)

[ 출력 25](#_Toc100824852)

[2.3.2. 시나리오 2 25](#_Toc100824853)

[ 입력 25](#_Toc100824854)

[ 출력 26](#_Toc100824855)

[2.3.3. 시나리오 3 28](#_Toc100824856)

[ 입력 28](#_Toc100824857)

[ 출력 28](#_Toc100824858)

[2.4. 보고서 작성 29](#_Toc100824859)

Ctrl을 누르면서 목차의 제목을 누르면 바로 이동합니다.

1. Interface

## 실습 목표

* 인터페이스의 개념과 그 용도 및 목적을 알아 본다.
* 인터페이스와 상속을 이용한 다형성을 이용하여 프로그램을 작성해 본다.

## 인터페이스 (interface)

### 인터페이스란?

인터페이스란 일반적으로 사람/시스템 간 서로 상호 작용하는 방법을 표준화 시킨 것을 뜻한다. 인터페이스는 이를 구현할 클래스(객체)가 어떤 operation을 수행할 수 있는지에 대해서는 정의하지만 그 operation을 어떻게 수행할 수 있는지는 정의하지 않는다.

### 자바의 인터페이스

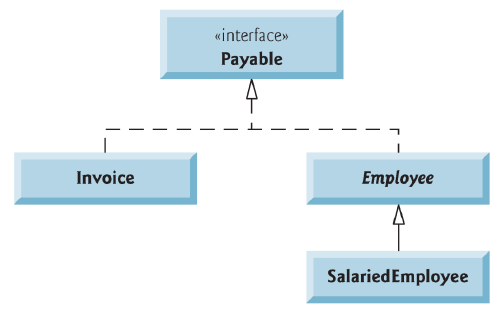
자바에서의 Interface는 interface 키워드로 선언할 수 있으며, 내부에 선언된 method들은 암시적으로 public abstract가, 내부에 선언된 field들은 암시적으로 public, static, final이 된다. (즉, constant와 abstract method만 존재할 수 있다.)

또한, Class와는 다르게 Interface member들은 반드시 public이어야 한다. interface 안에는 Default method만이 구현 코드이다. Default method를 제외한 어떠한 구현 코드도 포함하지 않는다. Interface를 사용하면 서로 상속관계가 없는 클래스들 사이에서도 공통의 method를 구현할 수 있다. Interface의 method를 구현하고자 하는 클래스들은 implements 키워드를 사용하여 interface의 method를 override한다. 또한 인터페이스를 구현하는 클래스는 해당 인터페이스가 가지고 있는 모든 메서드를 구현해야 한다.

**Abstract class와의 차이점:** 상속할 기본 구현이 없을 때 abstract class 대신 interface가 자주 사용된다.

### 인터페이스와 다형성

클래스가 타입으로 취급될 수 있었듯이 인터페이스도 타입으로 취급될 수 있다. 자바는 같은 인터페이스를 구현하는 두 클래스를 그 인터페이스의 타입으로 똑같이 취급할 수 있도록 해 주는 다형성을 제공한다. 단, 객체가 인터페이스 타입으로 취급될 때에는 그 인터페이스가 가지고 있는 메서드만 호출할 수 있다.



1. public interface Payable {
2. double getPaymentAmount();
3. }
4. public class Invoice implements Payable{
5. private int quantity;
6. private double pricePerItem;
8. public int getQuantity() { return quantity; }
9. public void setQuantity(int quantity) { this.quantity = quantity; }
10. public double getPricePerItem() { return pricePerItem; }
11. public void setPricePerItem(double pricePerItem) {
12. this.pricePerItem = pricePerItem;
13. }
15. @Override
16. public double getPaymentAmount() {
17. return getQuantity() \* getPricePerItem();
18. }
19. }
20. public abstract class Employee implements Payable{
21. private String name;
23. public Employee(String name) { super(); this.name = name; }
24. public String getName() { return name; }
25. public void setName(String name) { this.name = name; }
27. public abstract double earnings();
28. }
29. public class SalariedEmployee extends Employee {
30. private double weeklySalary;
32. public SalariedEmployee(String name) { super(name); }
33. public double getWeeklySalary() { return weeklySalary; }
34. public void setWeeklySalary(double weeklySalary) {
35. this.weeklySalary = weeklySalary;
36. }
38. @Override
39. public double earnings() {
40. return getWeeklySalary();
41. }
42. @Override
43. public double getPaymentAmount() {
44. return getWeeklySalary();
45. }
46. }

본 예제에서 Invoice 클래스와 Employee 클래스는 서로 상속 관계로 연결되지 않았지만, 같은 Payable 인터페이스를 구현하므로 둘을 Payable 객체로 똑같이 취급할 수 있다.

### default 인터페이스 메서드

인터페이스에서 default키워드로 선언된 메소드는 구현될 수 있다. 또한 인터페이스를 구현하는 클래스는 default메소드를 오버라이딩 하는 것도 가능하다.

1. public interface Calculator {
3. public int plus(int i, int j);
4. public int multiple(int i, int j);
6. default int exec(int i, int j){
7. //default로 선언함으로 메소드를 구현할 수 있다.
8. return i + j;
9. }
10. }
12. //Calculator인터페이스를 구현한 MyCalculator클래스
13. public class MyCalculator implements Calculator {
15. @Override
16. public int plus(int i, int j) {
17. return i + j;
18. }
20. @Override
21. public int multiple(int i, int j) {
22. return i \* j;
23. }
24. }
26. public class MyCalculatorExam {
28. public static void main(String[] args){
30. Calculator cal = new MyCalculator();
31. int value = cal.exec(5, 10);
32. System.out.println(value);
34. }
36. }

# 실습 과제

이번 과제에서는 실습 5주차의 게임 예제에 인터페이스를 도입하여 더 간결한 프로그램을 구현해 본다.



## 과제 제출 방법 (요약)

본 섹션에서는 과제 제출과 관련하여 지켜야 할 점을 요약 서술한다. 더 자세한 내용은 후술되는 장을 참고한다.

* 프로젝트 이름은 “Homework2\_자기학번”으로 생성한다.
  + 예시) 내 학번이 “202211111”인 경우 “Homework2\_202211111”로 프로젝트를 생성한다. (따옴표는 넣지 않는다.)
* 프로젝트를 export(내보내기)한 압축 파일의 이름도 “Homework2\_자기학번.zip”으로 지정한다.
  + 예시) 내 학번이 “202211111”인 경우 “Homework2\_202211111.zip”으로 압축파일을 생성한다. (따옴표는 넣지 않는다.)
  + 이클립스를 사용해 프로젝트 export를 할 경우 필히 classpath를 포함해야 한다. (포함하지 않았을 경우 import하는 쪽에서 제대로 열리지 않음.)
* 제출 기한은 2022년 4월 26일 23시 59분까지이다.
  + Export한 압축 파일과 실습 보고서를 모두 제출한다.
  + 지각 제출은 받지 않는다.
  + 보고서와 관련된 내용은 29쪽의 2.4장을 참고한다.

## 프로그램 구현

1. 배경

본 과제는 지난 실습 5주차의 주제였던 게임 애플리케이션을 확장하여 개발한다. 단, 인터페이스를 도입되어 공격 시스템과 스킬 시스템의 구조가 크게 변경되었고 과제의 복잡도를 낮추기 위하여 몇 가지 속성이 제거되었으므로 본 문서의 내용을 필히 확인한 후 구현에 착수할 수 있도록 한다.

실습 5주차의 과제에서는 공격 대상이 존재하지 않아 공격 시스템이 일종의 목업(mockup)의 형태로 구현되었지만 본 “과제 2”에서는 공격하여 부수면 공격 챔피언에게 골드를 보상으로 주는 보물 상자가 추가되었다. 기본적으로 챔피언이 공격하고 보물 상자가 피격되는 형태이지만 챔피언이 챔피언을 공격하는 경우도 발생할 수 있다.

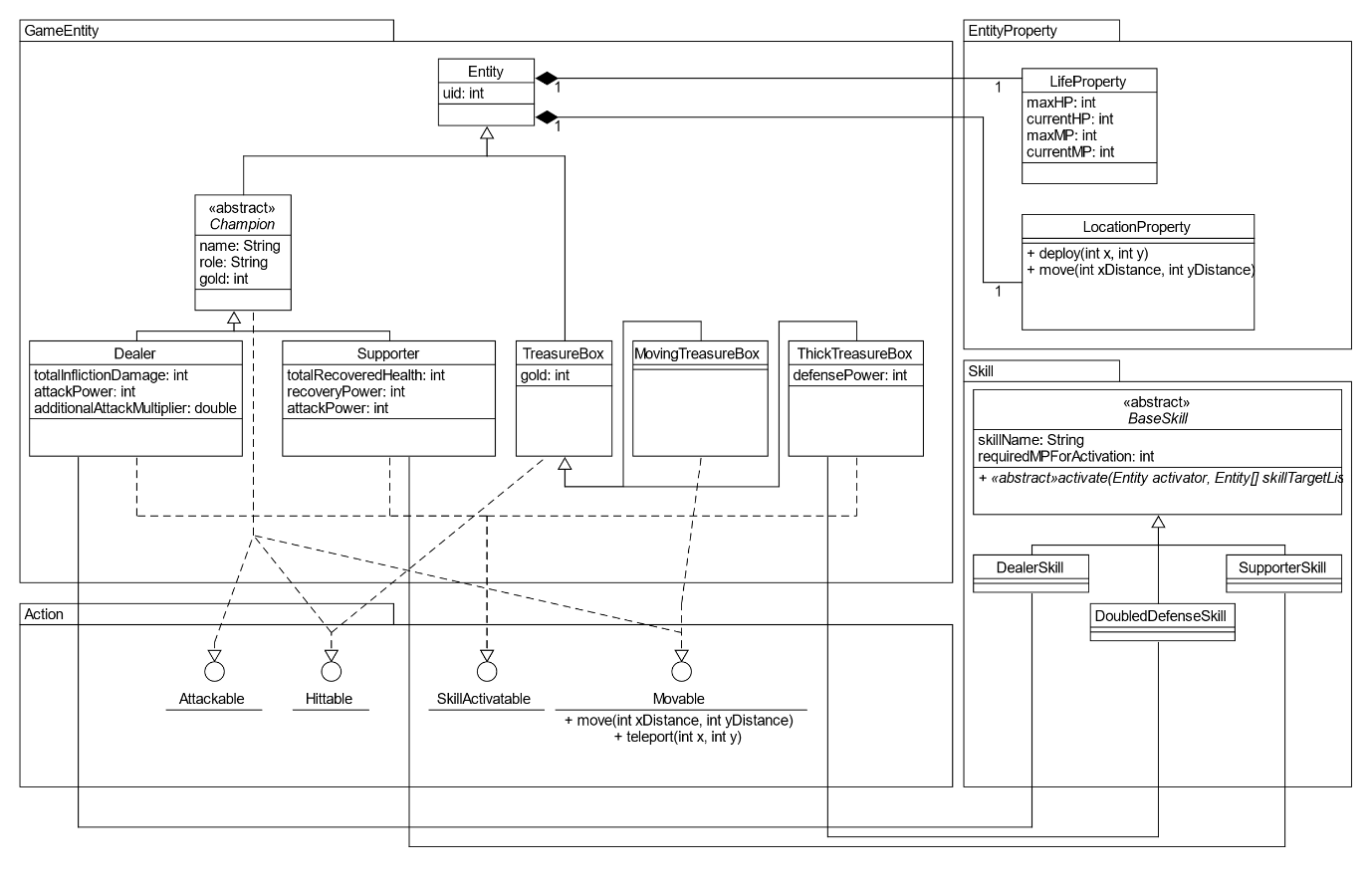
1. 모델링

이 장에서는 과제를 수행하기 위해 정의해야 하는 클래스 및 인터페이스 대해서 설명한다. 이 장에 지시되어 있는 클래스 이름, 필드 이름, 필드 메서드 이름, 인스턴스 이름, 인터페이스 이름, 인터페이스 메서드 이름 등은 지시되어 있을 경우 지시된 이름으로 정의해야 한다. 관계 또한 지시되어 있을 경우 지시된 대로 정의해야 한다. 따로 지시되어 있지 않은 요소의 경우 자유롭게 수정하거나 추가할 수 있다.

만약 따로 지시되어 있지 않을 경우…

* 필드 변수의 접근 제어자는 private 또는 protected 중 원하는 것으로 설정할 수 있다.
* 메서드의 접근 제어자는 private 또는 protected 또는 public 중 원하는 것으로 설정할 수 있다.

아래의 UML은 이번 과제에서 구현해야 하는 최소한의 애플리캐이션 구조를 나타내고 있다. 적어도 본 UML에 등장한 클래스, 인터페이스, 필드 변수, 메서드, 관계는 지켜져야 한다. 아래의 UML에서 클래스들을 그룹 지은 패키지는 비슷한 클래스들끼리 그룹 지은 것으로 실제 자바로 구현할 때 이 패키지대로 구분 지을 것인지는 자신의 디자인 철학에 따라도 된다.



본 그림은 벡터 이미지라 확대하면 상세히 볼 수 있다. MS 워드 상에서 확대가 제대로 되지 않을 경우 과제 탭에 별도로 첨부한 “과제 2클래스 다이어그램.svg”을 다운로드 받아 확인해도 된다.

* Entity 클래스

게임 내 모든 엔티티의 기본이 되는 클래스이다. Champion 클래스와 TreasureBox 클래스를 자식으로 두고 있다.

이 클래스는 아래의 필드를 가진다.

* uid
  + 정수형 값으로 각 엔티티를 구분짓는 고유 id이다. 엔티티의 id는 프로그램의 입력으로 주어지므로 해당 값을 받아 저장할 수 있으면 된다.

이 클래스는 LifeProperty 클래스와 LocationProperty 클래스를 composition 관계로 가지고 있다. 즉, Entity 클래스와 나머지 두 LifeProperty, LocationProperty는 라이프 사이클이 같다.

* Champion 클래스

게임 내 챔피언들의 기본이 되는 클래스이다. Entity 클래스를 상속받으며 Dealer와 Supporter 클래스를 자식으로 두고 있다.

이 클래스는 추상 클래스이다.

이 클래스는 아래의 필드를 가진다.

* name
  + String 값으로 챔피언 엔티티의 이름을 나타낸다. 이름은 챔피언 엔티티가 생성될 때 프로그램 입력으로 주어진다.
  + 한 번 설정된 이름이 변경되는 경우는 없다.
* role
  + String 값으로 해당 챔피언이 어떤 속성의 챔피언인지를 나타낸다.
  + 챔피언 인스턴스가 어떤 클래스인지에 따라 값이 다르다.
    - Dealer의 경우 “dealer”의 값으로 설정된다.
    - Supporter의 경우 “supporter”의 값으로 설정된다.
* gold
  + 정수형 값으로 현재 챔피언이 가지고 있는 소지금을 나타낸다.
  + 소지금은 전투를 통해 획득할 수 있다. 전투에 대한 내용은 19쪽의 “공격 처리”를 참고한다.
  + 초기값은 0이다.

이 클래스는 아래의 인터페이스를 구현한다.

* Attackable
* Hittable
* Movable
* Dealer 클래스

공격에 특화된 챔피언을 나타내는 클래스이다. Champion을 상속받는다.

이 클래스는 아래의 필드 변수를 가진다.

* totalInflictionDamage
  + 정수형 값으로 현재까지의 누적 데미지를 나타낸다.
  + 누적 데미지는 공격이 성공했을 때 자신이 가한 “공격 데미지”의 양만큼 증가시킨다. 이 내용과 관련된 자세한 내용은 19쪽의 “공격 처리”를 참고한다.
    - 여러가지 사유로 공격이 무효되었을 때는 증가하지 않는다. 공격이 무효될 수 있는 상황에 대해서는 23쪽의 “특수 처리”를 참고한다.
    - 이 값은 일반 공격으로 인한 공격 데미지만 가산되며 스킬 공격으로 인한 공격 데미지는 가산되지 않는다. 이 내용은 16쪽의 DealerSkill 클래스 설명에서 다시 확인할 수 있다.
* attackPower
  + 정수형 값으로 자신의 기본 공격력이다.
* additionalAttackMultiplier
  + 실수형 값으로 공격 및 스킬에서 최종 데미지를 계산하기 위해 사용되는 곱셈 연산용 숫자이다.

이 클래스는 아래의 클래스의 객체를 필드 변수로 가진다. (association)

* DealerSkill
  + 이 클래스와 연관되는 DealerSkill 객체는 name이 “moreAttack!!!”이고 발동에 필요한 MP가 50인 skill 객체이여야 한다. 15쪽의 BaseSkill 클래스와 함께 보도록 한다.

이 클래스는 아래의 인터페이스를 구현한다.

* SkillActivatable
* Supporter 클래스

지원에 특화된 챔피언을 나타내는 클래스이다. Champion을 상속받는다. 이 엔티티는 공격했을 때 자신을 회복하는 능력을 가지고 있다.

이 클래스는 아래의 필드 변수를 가진다.

* totalRecoveredHealth
  + 정수형 값으로 현재까지의 자신의 공격으로 인한 총 회복량을 의미한다.
    - 스킬로 인한 회복은 집계되지 않는다. 자신의 공격으로만 인한 회복만 집계한다.
  + 이 클래스는 공격이 성공했을 때 아래의 수식만큼 자신의 회복을 시도하고 시도한 양을 totalRecoveredHealth에 더한다.
    - 회복량은 기본적으로 정수형이다. 계산된 회복량이 실수형일 경우 소수점 이하를 버려 정수형으로 만든다.
* recoveryPower
  + 정수형 값으로 자신의 기본 회복력이다.
* attackPower
  + 정수형 값으로 자신의 기본 공격력이다.

이 클래스는 아래의 클래스의 객체를 필드 변수로 가진다. (association)

* SupporterSkill
  + 이 클래스와 연관되는 SupporterSkill 객체는 name이 “allHeal”이고 발동에 필요한 MP가 100인 skill 객체이여야 한다. 15쪽의 BaseSkill 클래스와 함께 보도록 한다.

이 클래스는 아래의 인터페이스를 구현한다.

* SkillActivatable
* TreasureBox 클래스

챔피언들의 공격 대상이 되는 엔티티이다. 골드를 가지고 있다. 이 골드는 나중에 TreasureBox 엔티티의 HP을 0으로 만드는 데에 성공한 챔피언에게 지급되는 데에 사용된다. 골드 지급에 대한 내용은 19쪽의 “공격 처리”를 참고한다. 이 클래스는 자식으로 MovingTreasureBox와 ThickTreasureBox를 두고 있다.

이 클래스는 아래의 필드 변수를 가진다.

* gold
  + 정수형 값으로 이 객체가 가지고 있는 소지 금화의 값을 나타낸다. 나중에 챔피언에게 지급되고 나면 0이 된다.

이 클래스는 아래의 인터페이스를 구현한다.

* Hittable
* MovingTresaureBox 클래스

위치 이동이 가능한 보물 상자이다. TreasureBox를 상속받는다.

이 클래스는 아래의 인터페이스를 구현한다.

* Movable
* ThickTreasureBox 클래스

단단한 보물 상자를 의미한다. 이 보물 상자는 자신에게 들어온 데미지를 자신의 방어력만큼 경감할 수 있는 능력을 가지고 있다. 이 클래스는 TreasureBox를 상속받는다.

이 클래스는 아래의 필드 변수를 가진다.

* defensePower
  + 정수형 값으로 자신의 방어력을 나타낸다. 자신에게 들어온 데미지는 먼저 이 방어력 만큼 경감된 다음 자신의 HP에 반영된다. 더 자세한 내용은 19쪽의 “공격 처리”를 참고한다.

이 클래스는 아래의 인터페이스를 구현한다.

* SkillActivatable

이 클래스는 아래의 클래스의 객체를 필드 변수로 가진다. (association)

* DoubledDefenseSkill
  + 이 클래스와 연관되는 DoubledDefenseSkill 객체는 name이 “doubleDefense”이고 발동에 필요한 MP가 20인 skill 객체이여야 한다. 15쪽의 BaseSkill 클래스와 함께 보도록 한다.
* LifeProperty 클래스

이 클래스는 Entity의 생명과 관련된 수치와 그 처리를 담당한다. 모든 Entity 객체는 LifeProperty 객체를 1개 가지고 있다. 이 클래스는 Entity 클래스와 composition 관계이며 Entity 클래스가 이 클래스를 소유한다.

이 클래스는 다음의 필드 변수를 가진다.

* maxHP
  + 정수형 값으로 currentHP의 최대값이다. currentHP가 이 값을 넘으려고 하는 경우 이 값으로 바운딩된다.
* currentHP
  + 정수형 값으로 엔티티의 현재 체력을 나타낸다.
  + 이 값은 0 이상 maxHP 이하이다.
  + 이 값이 0 미만으로 내려가려고 하는 경우 0으로 바운딩된다.
  + 이 값이 maxHP 초과하려고 하는 경우 maxHP로 바운딩된다.
* maxMP
  + 정수형 값으로 currentMP의 최대값이다. currentMP가 이 값을 넘으려고 하는 경우 이 값으로 바운딩된다.
* currentMP
  + 정수형 값으로 엔티티의 현재 마력을 나타낸다.
  + 이 값은 0 이상 maxMP 이하이다.
  + 이 값이 0 미만으로 내려가려고 하는 경우 0으로 바운딩된다.
  + 이 값이 maxMP 초과하려고 하는 경우 maxMP로 바운딩된다.
* LocationProperty 클래스

이 클래스는 Entity의 위치와 관련된 수치와 그 처리를 담당한다. 모든 Entity 객체는 LocationProperty 객체를 1개 가지고 있다. 이 클래스는 Entity 클래스와 composition 관계이며 Entity 클래스가 이 클래스를 소유한다.

이 클래스는 다음의 메서드를 제공한다.

* public deploy(int x, int y)
  + 엔티티의 위치를 (x, y)로 이동시킨다.
* public move(int xDistance, int yDistance)
  + 엔티티의 위치를 현재 위치에서 x축으로 xDistance만큼 y축으로 yDistance만큼 이동시킨다.
* BaseSkill 클래스

이 클래스는 모든 스킬의 부모 클래스로 스킬의 기본적인 속성을 정의한다. 이 클래스는 자식으로 DealerSkill, SupporterSkill, DoubledDefenseSkill 클래스를 두고 있다.

이 클래스는 추상 클래스이다.

이 클래스는 아래의 필드 변수를 가진다.

* skillName
  + String 값으로 이 스킬의 이름을 지정한다. 스킬의 이름은 나중에 어떤 스킬을 발동시킬 지 결정할 때 스킬 식별자로서 사용된다.
* requiredMPForActivation
  + 정수형 값으로 이 스킬을 발동하기 위해 엔티티가 지불해야 하는 MP를 나타낸다. 스킬이 발동한 경우 엔티티의 currentMP(15쪽의 LifeProperty 참고)는 이 수치만큼 감소한다.
  + 이 스킬을 발동하려는 엔티티가 이 수치만큼의 MP를 가지고 있지 않은 경우 스킬 발동은 무효된다. 무효와 관련된 처리는 23쪽의 “특수 처리”를 참고한다.

이 클래스는 아래의 메서드를 제공한다.

* public abstract activate(Entity activator, Entity[] skillTargetList)
  + 추상 메서드이다.
  + 스킬 발동 시 이 메서드가 호출된다.
  + 이 메서드가 호출되면 이 스킬의 처리를 진행한다.
  + activator는 이 스킬을 발동한 엔티티를 나타낸다.
  + skillTargetList는 이 스킬의 대상이 되는 엔티티들을 나타낸다.
    - 스킬 대상은 1개 또는 1개보다 많을 수 있으며 어떤 스킬은 대상으로 하는 객체가 자기 자신만으로 skillTargetList는 0개인 스킬도 있다.
* DealerSkill 클래스

딜러용 스킬을 기술한다. 이 클래스는 BaseSkill 클래스를 상속받는다.

이 스킬의 발동 처리로서 스킬의 대상이 된 엔티티(1개)에게 아래만큼의 데미지로 공격한다.

totalInflictionDamage와 additionalAttackMultipliers는 Dealer 클래스의 속성으로서 스킬 발동자의 값을 사용한다. 공격 데미지가 정수형이 아닌 경우 소수점 이하를 버려 정수형으로 만든다. 공격 처리에 대한 더 자세한 내용은 19쪽의 “공격 처리”를 확인한다.

이 스킬은 다음과 같은 사유로 발동이 무효가 될 수 있다. 무효와 관련된 처리는 23쪽의 “특수 처리”를 참고한다.

* 스킬 발동자의 MP가 부족
* 스킬 발동자가 Dealer 객체가 아님
* 스킬 대상자가 Hittable 객체가 아님
* 스킬 대상자가 1개가 아님.

딜러는 매 공격마다 자신이 가한 공격 데미지를 가산하지만 스킬로 인한 공격 데미지는 가산하지 않는다. 이 내용은 11쪽의 Dealer 클래스 설명에서도 확인할 수 있다.

* SupporterSkill 클래스

서포터용 스킬을 기술한다. 이 클래스는 BaseSkill 클래스를 상속받는다.

이 스킬의 발동 처리로서 스킬의 대상이 된 모든 엔티티에게 아래만큼 회복시킨다.

totalRecoverdHealth와 recoveryPower는 Supporter 클래스의 속성으로서 스킬 발동자의 값을 사용한다. 최종 산정 회복량이 정수형이 아닌 경우 소수점 이하를 버려 정수형으로 만든다. 스킬 대상자 목록에는 스킬 발동자가 포함될 수도 있다.

이 스킬은 다음과 같은 사유로 발동이 무효가 될 수 있다. 무효와 관련된 처리는 23쪽의 “특수 처리”를 참고한다.

* 스킬 발동자의 MP가 부족
* 스킬 발동자가 Supporter 객체가 아님
* 스킬 대상자 중 Champion 객체가 아닌 것이 있음.
  + 이 경우 스킬 자체가 발동되지 않으므로 스킬 대상자에 Champion 객체와 Champion 객체가 아닌 것이 섞여 있다 하더라도 대상이 된 Champion 객체의 체력이 상승하면 안 된다.
* DoubledDefenseSkill 클래스

단단한 보물 상자용 스킬을 기술한다. 이 클래스는 BaseSkill 클래스를 상속받는다.

이 스킬의 발동 처리로서 스킬 발동자의 방어력(defensePower)은 기존값의 2배가 된다. 이 스킬은 대상자가 없다. defensePower는 스킬 발동자의 값을 사용한다.

이 스킬은 다음과 같은 사유로 발동이 무효가 될 수 있다. 무효와 관련된 처리는 23쪽의 “특수 처리”를 참고한다.

* 스킬 발동자의 MP가 부족
* 스킬 발동자가 ThickTreasureBox가 아님
* 스킬 대상자가 1개 이상 지정됨.
* Attackable 인터페이스

이 인터페이스는 공격을 개시할 수 있는 모든 엔티티가 구현하는 인터페이스이다.

이 인터페이스를 구현하는 클래스는 다음과 같다.

* Champion
* Hittable 인터페이스

이 인터페이스는 공격의 대상이 될 수 있는 모든 엔티티가 구현하는 인터페이스이다.

이 인터페이스를 구현하는 클래스는 다음과 같다.

* Champion
* TreasureBox
* Movable 인터페이스

이 인터페이스는 이동을 할 수 있는 모든 엔티티가 구현하는 인터페이스이다. 이 인터페이스를 구현하지 않는 클래스들은 최초 위치에서 이동하지 못한다.

이 인터페이스는 아래의 메서드를 제공한다.

* public move(int xDistance, int yDistance)
  + 15쪽의 LocationProperty의 move 메서드와 같은 역할을 한다.
* public teleport(int x, int y)
  + 15쪽의 LocationProperty의 deploy 메서드와 같은 역할을 한다.

이 인터페이스를 구현하는 클래스는 다음과 같다.

* Champion
* MovingTreasureBox

이 클래스를 구현하지 않는 엔티티를 대상으로 하는 이동 명령은 무효가 된다. 무효 처리에 대한 자세한 내용은 23쪽의 “특수 처리”를 참고한다.

* SkillActivatable 인터페이스

이 인터페이스는 스킬을 사용할 수 있는 모든 엔티티가 구현하는 인터페이스이다. 본 과제에서 이 인터페이스를 구현하는 클래스는 다음과 같다.

* Dealer
* Supporter
* ThickTreasureBox

1. 공격 처리

이 장에서는 공격의 처리 순서를 기술한다.

공격은 공격자 1명과 피격자 1명이 존재하는 1:1 관계이다. 누가 공격자가 되고 누가 피격자가 되는지는 프로그램의 입력으로 주어진다. 공격자와 피격자가 선정되고 나면 공격 처리가 진행된다. 공격은 아래와 같은 순서로 처리된다.

1. 공격자는 자신이 얼만큼의 “데미지”로 피격자를 공격할 것인지 그 데미지를 계산한다.
   * 데미지는 기본적으로 정수형이다. 데미지가 최종적으로 실수형으로 도출된 경우 소수점 이하를 버려 정수형으로 만든다.
   * 데미지는 0보다 작을 수 없다. 0보다 작을 경우 0으로 바운딩 된다.
2. 피격자에게 계산된 데미지를 “공격 데미지”로서 가한다.
3. 피격자는 자신에게 가해진 “공격 데미지”의 수치를 조정하여 “최종 산정 데미지”를 계산한다.
   * ThickTreasureBox의 defensePower는 이 시점에서 적용된다.
   * 최종 산정 데미지는 기본적으로 정수형이다. 데미지가 최종적으로 실수형으로 도출된 경우 소수점 이하를 버려 정수형으로 만든다.
   * 최종 산정 데미지는 0보다 작을 수 없다. 0보다 작을 경우 0으로 바운딩 된다.
4. “최종 산정 데미지”만큼 피격자의 HP가 하락한다.
5. HP가 0이 된 경우 해당 객체를 사망 취급하고 자신의 gold를 전부 공격자에게 지급한다.
   * gold를 지급하였으므로 피격자의 gold는 0이 되어야 한다.
6. 이후 공격자와 피격자 각각이 처리할 것에 대하여 처리한다.
   * 예를 들어 Supporter 클래스의 경우 자신의 회복 처리를 진행해야 하며 Dealer 클래스의 경우 자신의 누적 데미지를 증가시켜야 한다.
   * Dealer 클래스가 자신의 누적 데미지에 더해야 할 것은 “공격 데미지”이지 “최종 산정 데미지”가 아님에 유의한다. 또한 스킬로 인한 공격일 경우에는 가산하지 않는다.

발동 시 처리로서 공격이 포함된 스킬의 경우에도 공격 처리는 위와 같다.

Attackable을 구현하는 클래스들의 “공격 데미지” 계산 방식은 아래와 같다.

* Dealer
* Supporter

Hittable을 구현하는 클래스들의 “최종 산정 데미지” 계산 방식은 아래와 같다.

* ThickTreasureBox
* ThickTreasureBox를 제외한 그 외 나머지 클래스

공격 명령은 다음과 같은 사유들로 인해 무효가 될 수 있다. 무효와 관련된 처리는 23쪽의 “특수 처리”를 참고한다.

* 공격자가 Attackable이 아님
* 피격자가 Hittable이 아님
* 공격자가 이미 사망함.
* 피격자가 이미 사망함.

1. 프로그램의 입력

프로그램의 입력은 표준 입력 스트림[[1]](#footnote-1)으로 한 줄 한 줄 입력된다. 명령은 총 다섯 종류가 있다.

명령어의 [] 부분에 값이 들어온다.

* create 명령
  + 엔티티를 생성하는 명령이다. 엔티티 종류에 따라 포맷이 다르다.
  + Dealer
    - create Dealer [uid] [이름] [HP] [MP] [x] [y] [attackPower] [additionalAttackMultiplier]
  + Supporter
    - create Supporter [uid] [이름] [HP] [MP] [x] [y] [attackPower] [recoveryPower]
  + TreasureBox
    - create TreasureBox [uid] [HP] [MP] [x] [y] [gold]
  + MovingTreasureBox
    - create MovingTreasureBox [uid] [HP] [MP] [x] [y] [gold]
  + ThickTreasureBox
    - create ThickTreasureBox [uid] [HP] [MP] [x] [y] [gold] [defensePower]
  + HP와 MP는 maxHP와 maxMP를 의미하며 동시에 currentHP와 currentMP를 maxHP와 maxMP로 맞춘다.
  + uid는 중복되지 않게 입력된다. 따로 중복 검사할 필요는 없다.
* move 명령
  + 지정된 엔티티를 현재 위치에서 주어진 x와 y 변위만큼 이동시킨다.
  + 포맷은 아래와 같다.
    - move [mover uid] [xDistance] [yDistance]
* attack 명령
  + 공격자로 지정된 엔티티가 피격자로 지정된 엔티티를 공격하는 명령이다.
  + 포맷은 아래와 같다.
    - attack [attacker uid] [defender uid]
  + 자기 자신을 공격하는 예시는 주어지지 않는다.
* skill 명령
  + skill를 발동시키는 명령어이다.
  + 포맷은 다음과 같다.
    - skill [activator uid] [activated skill name] [skill target uid 1] [skill target uid 2] [skill target uid 3] ...
  + 발동자는 1명이며 발동 스킬의 이름이 주어진다.
  + 대상자는 0명 이상이다. 없을 수도 있고 1명일 수도 있으며 여러 명일 수도 있다.
* end
  + 프로그램 상태를 출력하고 프로그램을 종료한다. 후술되는 프로그램 출력 부분을 참고한다.
  + 포맷은 다음과 같다.
    - end

1. 프로그램 출력

이 장에서는 프로그램의 출력을 지정한다. 본 과제는 프로그램을 이용하여 출력을 자동 채점할 것이므로 이 장에서 지정하는 프로그램 출력 포맷을 반드시 지켜야 한다.

end 명령어가 입력되면 콘솔 창 마지막에 모든 엔티티의 정보를 출력해야 한다. 각 엔티티가 한 줄씩 출력하며, 출력 순서는 프로그램에서 생성된 순서대로 출력한다. 즉 먼저 생성된 엔티티가 출력도 먼저 한다.

각 엔티티별 출력해야 하는 정보와 그 포맷은 아래와 같다. 아래의 포맷에서 {}으로 감싸진 부분을 해당 정보로 치환해서 출력해야 한다. 중괄호를 포함하여 치환한다. 예시는 후술되는 프로그램 예시 부분을 참고하면 된다.

* Dealer
  + [Entity {uid}] [location ({x}, {y})] [Life HP: {HP} MP: {MP}] [Champion {name} {role} (gold: {gold})] [Dealer (total infliction damage: {total infliction damage})]
* Supporter
  + [Entity {uid}] [location ({x}, {y})] [Life HP: {HP} MP: {MP}] [Champion {name} {role} (gold: {gold})] [Supporter (total recovered health: {total recovered health})]
* TreasureBox
  + [Entity {uid}] [location ({x}, {y})] [Life HP: {HP} MP: {MP}] [TreasureBox (gold: {gold})]
* MovingTreasureBox
  + [Entity {uid}] [location ({x}, {y})] [Life HP: {HP} MP: {MP}] [TreasureBox (gold: {gold})] [MovingTreasureBox]
* ThickTreasureBox
  + [Entity {uid}] [location ({x}, {y})] [Life HP: {HP} MP: {MP}] [TreasureBox (gold: {gold})] [ThickTreasureBox (defense power: {defensePower})]

1. 특수 처리

이 장에서는 프로그램 구현 상 지켜야할 규칙들을 요약하여 정리한다. 아래의 규칙에서 “무효가 된다”는 것은 명령의 실행이 취소되고 게임의 상태가 변경되지 않아야 한다는 것으로 그 명령 자체가 입력되지 않은 것처럼 처리해야 함을 의미한다. 즉, 무효가 되는 명령은 그냥 무시되고 다음 명령을 처리해야 한다.

* 존재하지 않는 uid를 대상으로 하는 모든 명령은 무효가 된다.
* HP가 0 이하인 엔티티는 사망 상태로 취급한다.
  + 사망 상태의 엔티티의 HP가 회복되어 0보다 커질 경우 사망 상태는 취소되고 다시 일반 엔티티 취급을 받게 된다.
    - 즉, 사망 상태의 엔티티도 체력 회복이 가능하다.
* 사망 상태의 엔티티를 대상으로 하는 공격은 무효가 된다.
  + 공격 스킬 역시 사망 상태를 대상으로 하는 스킬일 경우 발동이 무효 된다.
* 사망 상태의 엔티티가 개시하는 공격은 무효가 된다.
* 자기 자신을 공격하는 예시는 주어지지 않는다.
* 사망 상태의 엔티티가 발동하는 스킬은 무효가 된다.
* 스킬 발동자가 대상으로 하는 스킬을 발동하는 데에 필요한 MP를 소지하지 않은 경우 스킬 발동은 무효가 된다.
* 사망 상태의 엔티티라도 이동은 가능하다.
* 두 엔티티는 같은 위치에 존재할 수 있다.
  + 엔티티가 위치할 수 없는 위치는 존재하지 않는다. 맵의 경계는 없다.
* 공격은 공격 개시자와 피격 대상자의 거리의 상관없이 개시할 수 있다.
* 스킬은 스킬 발동자와 스킬 대상자들의 거리에 상관없이 발동할 수 있다.

## 프로그램 실행 예시

입력은 전체를 복사해서 터미널에 붙여 넣으면 쉽게 입력할 수 있다. 아래의 출력에서는 어느 시점에서 어떤 명령어가 무효가 되는지를 보여주기 위해 명령의 처리 과정이 출력에 포함되어 있지만 이걸 구현할 필요는 없고 프로그램이 출력할 부분은 빨간색으로 처리된 부분이다. 프로그램 출력 내 해당 내용이 있으면 된다.

### 시나리오 1

* 입력

|  |
| --- |
| create Dealer 1 amo 100 100 0 0 10 0.2  create TreasureBox 2 40 100 1 1 100  attack 1 2  attack 1 2  move 1 2 3  attack 2 1  attack 1 2  attack 1 2  attack 1 2  attack 1 2  attack 1 2  end |

* 출력

|  |
| --- |
| ---명령 순차 처리----  create Dealer 1 amo 100 100 0 0 10 0.2  create TreasureBox 2 40 100 1 1 100  attack 1 2  attack 1 2  move 1 2 3  attack 2 1  The attacker can not attack.  attack 1 2  attack 1 2  attack 1 2  invalidated.  attack 1 2  invalidated.  attack 1 2  invalidated.  end  -----  [Entity 1] [location (2, 3)] [Life HP: 100 MP: 100] [Champion amo dealer (gold: 100)] [Dealer (total infliction damage: 48)]  [Entity 2] [location (1, 1)] [Life HP: 0 MP: 100] [TreasureBox (gold: 0)]  ---end of program--- |

### 시나리오 2

* 입력

|  |
| --- |
| create Dealer 1 amo 100 150 0 0 10 0.5  create Supporter 2 popo 100 200 10 5 10 2  create MovingTreasureBox 3 50 100 8 7 100  create ThickTreasureBox 4 50 100 7 4 200 5  move 4 1 1  attack 2 3  attack 2 3  attack 2 3  attack 2 3  attack 1 3  attack 2 3  move 3 -8 -7  attack 2 1  attack 2 1  attack 2 1  attack 2 1  attack 2 1  attack 2 1  attack 2 1  attack 2 1  attack 2 1  attack 2 1  attack 1 4  skill 2 allHeal 1 2  attack 1 4  skill 1 moreAttack!!! 4  attack 1 4  attack 1 4  skill 1 moreAttack!!! 3  skill 1 moreAttack!!! 4  end |

* 출력

|  |
| --- |
| ---명령 순차 처리----  create Dealer 1 amo 100 150 0 0 10 0.5  create Supporter 2 popo 100 200 10 5 10 2  create MovingTreasureBox 3 50 100 8 7 100  create ThickTreasureBox 4 50 100 7 4 200 5  move 4 1 1  It is not movable.  attack 2 3  attack 2 3  attack 2 3  attack 2 3  attack 1 3  attack 2 3  invalidated.  move 3 -8 -7  attack 2 1  attack 2 1  attack 2 1  attack 2 1  attack 2 1  attack 2 1  attack 2 1  attack 2 1  attack 2 1  attack 2 1  attack 1 4  inactivated.  skill 2 allHeal 1 2  attack 1 4  skill 1 moreAttack!!! 4  attack 1 4  attack 1 4  skill 1 moreAttack!!! 3  invalidated.  skill 1 moreAttack!!! 4  end  -----  [Entity 1] [location (0, 0)] [Life HP: 8 MP: 50] [Champion amo dealer (gold: 200)] [Dealer (total infliction damage: 60)]  [Entity 2] [location (10, 5)] [Life HP: 100 MP: 100] [Champion popo supporter (gold: 100)] [Supporter (total recovered health: 42)]  [Entity 3] [location (0, 0)] [Life HP: 0 MP: 100] [TreasureBox (gold: 0)] [MovingTreasureBox]  [Entity 4] [location (7, 4)] [Life HP: 0 MP: 100] [TreasureBox (gold: 0)] [ThickTreasureBox (defense power: 5)]  ---end of program--- |

### 시나리오 3

* 입력

|  |
| --- |
| create Dealer 1 mac 200 200 8 7 20 0.5  create ThickTreasureBox 42 35 100 0 2 200 10  attack 1 42  skill 42 doubleDefense  attack 1 42  skill 42 doubleDefense  attack 1 42  skill 1 moreAttack!!! 42  skill 42 doubleDefense  skill 42 doubleDefense  end |

* 출력

|  |
| --- |
| ---명령 순차 처리----  create Dealer 1 mac 200 200 8 7 20 0.5  create ThickTreasureBox 42 35 100 0 2 200 10  attack 1 42  skill 42 doubleDefense  attack 1 42  skill 42 doubleDefense  attack 1 42  skill 1 moreAttack!!! 42  skill 42 doubleDefense  invalidated.  skill 42 doubleDefense  invalidated.  end  -----  [Entity 1] [location (8, 7)] [Life HP: 200 MP: 150] [Champion mac dealer (gold: 200)] [Dealer (total infliction damage: 90)]  [Entity 42] [location (0, 2)] [Life HP: 0 MP: 60] [TreasureBox (gold: 0)] [ThickTreasureBox (defense power: 40)]  ---end of program--- |

## 보고서 작성

보고서에는 프로그램 실행 예시의 시나리오 1과 2를 실행한 결과를 각각 첨부하여 제출하고 자신의 공격 처리 구현에 대해 간략하게 서술한다.

시나리오 3은 첨부하지 않는다.

1. 콘솔로 입력된다는 뜻이다. [↑](#footnote-ref-1)