# 창의적 소프트웨어 설계 실습 문제 06 - hw06-3

**제출 기한**

- 10월 22일 월 23:59까지

1. hw6-3(mkdir hw6-3)라는 폴더를 만들고 GitLab에 push
2. hw6-3 디렉토리에 Makefile과 각 문제에서 요구하는 파일들을 작성
3. **‘make’명령을 수행하여 숙제가 빌드**
4. 최종 버전을 GitLab에 commit
5. **시간과 파일명, 입력과 출력 방식 반드시 지키기**

## 과제1. Minimal Fighter

MinimalFighter class 작성

- private 멤버 변수

int mHp (체력)

int mPower (공격력)

FighterStatus mStatus (상태: Invalid, Alive, Dead - defined by enum)

- 생성자

MinimalFighter() : 체력 0, 공격력 0, 상태 ‘Invalid’로 초기화

MinimalFighter(int \_hp, int \_power) : 체력과 공격력을 주어진 값으로, 상태 ‘Alive’나 ‘Dead’로

초기화 (아래의 규칙 참조)

- 멤버 함수

int hp() const

int power() const

FighterStatus status() const : 체력, 공격력, 상태를 리턴

void setHp(int \_hp) : 체력을 변경

void hit(MinimalFighter \*\_enemy) : Hit 명령 수행 (아래의 규칙 참조)

void attack(MinimalFighter \*\_target) : Attack 명령 수행 (아래의 규칙 참조)

void fight(MinimalFighter \*\_enemy) : Fight 명령 수행 (아래의 규칙 참조)

- Minimal Fighter 규칙

1. 모든 Fighter는 체력이 0 이하로 떨어지면 죽으며 (status = Dead), 입력된 체력이 0 이하일 경우는 이미 죽어있는 상태라 아래의 모든 행동을 할 수 없다.
2. Hit - Fighter가 enemy와 공격을 동시에 한 번씩 교환한다. 공격을 하면 공격력만큼 상대방의 체력이 감소된다.
3. Attack - Fighter가 target을 일방적으로 공격한다. target의 체력이 공격력만큼 감소하며, Fighter의 공격력이 0이 된다. Fighter의 체력은 변화가 없다.
4. Fight - Fighter와 enemy가 둘 중 하나가 죽을 때까지 공격을 교환한다. 동시에 죽을 경우 둘 다 죽은 것으로 처리한다.
5. 예시  
   H2, P1 hit H1, P1 -> H1, P1 / DEAD (H0, P1)  
   H3, P1 hit H3, P1 -> H2, P1 / H2, P1  
   H1, P3 attack H3, P1 -> H1, P0 / DEAD  
   H4, P1 fight H3, P1 -> H3, P1 / H2, P1 -> H2, P1 / H1, P1 -> H1, P1 / DEAD

작성 규칙

* 주어진 skeleton에서 MinimalFighter 클래스를 구현한다. (인터페이스, 메인함수 부분은 수정하지 않음)
* 위의 규칙을 모두 반영할 수 있도록 구현한다.
* 올바르지 않은 입력이 들어온 경우 종료.

파일명 : minimal\_fighter  
 (minimal\_fighter.cc, minimal\_fighter.h, minimal\_fighter\_main.cc)

입력 : 양의 정수나 0인 Fighter1의 체력과 공격력, 액션(H, A, F), 양의 정수나 0인 Fighter2의 체력과 공격력

출력 : 액션 이후 두 Fighter의 상태를 출력 (fight의 경우 최종 결과만 출력)

|  |
| --- |
| **$ ./minimal\_fighter**  **2 1 H 1 1**  **H1, P1 / DEAD**  **3 1 H 3 1**  **H2, P1 / H2, P1**  **1 3 A 3 1**  **H1, P0 / DEAD**  **4 1 F 3 1**  **H1, P1 / DEAD**  **q**  **$** |

## 과제 2. 계좌 클래스 구현

* 계좌 클래스는 계좌 주인의 ID와 잔고를 멤버변수로 갖는다
* 계좌 관리자 클래스는 계좌 클래스와 관리하는 계좌 개수를 멤버변수로 갖는다
* 계좌 관리자는 입금, 출금, 이체, 잔고확인 함수를 갖는다
* 계좌가 보유할 수 있는 최대 금액은 100만원, 최저 금액은 0원이며, 1회 입금/출금/이체 한도 금액은 없다
* 한도 초과로 입금/출금/이체가 불가능한 경우의 예외처리 필요(이체는 양쪽에서 확인)
* 각 작업 이후에는 잔고를 표기한다
* 처음 생성된 계좌의 잔고는 반드시 0원이다
* 계좌는 최대 10개까지 생성 가능하며, ID는 계좌 생성 순서대로 0~9가 할당된다
* 10개 초과 계좌 생성에 대한 예외처리 필요
* 명령어 오타, type error 등은 예외처리하지 않는다.

파일명: simple\_account (simple\_account.h, simple\_account.cc, simple\_account\_main.cc)

입력 :

* 입력은 네가지 형태가 가능하다.
* 계좌 생성을 위해서는 1개의 입력 (N)을 제공한다.
* 입금을 위해서는3개의 입력 (D, 계좌ID, 입금금액)을 제공한다
* 출금을 위해서는 3개의 입력 (W, 계좌ID, 입금금액)을 제공한다
* 이체를 위해서는 4개의 입력 (T, 이체하려는 계좌 ID (from), 이체 대상이 되는 계좌 ID (to), 금액)을 제공한다.
* 마치려면 Q를 입력한다.

출력 :

|  |
| --- |
| **$ ./simple\_account**  **Job?**  **D 0 50000**  **Account does not exist**  **Job?**  **N**  **Account for user 0 registered**  **Balance of user 0: 0**  **Job?**  **D 0 50000**  **Success: Deposit to user 0 50000**  **Balance of user 0: 50000**  **Job?**  **W 0 100000**  **Failure: Withdraw from user 0 100000**  **Balance of user 0: 50000**  **Job?**  **N**  **Account for user 1 registered**  **Balance of user 1: 0**  **Job?**  **T 1 0 30000**  **Failure: Transfer from user 1 to user 0 30000**  **Balance of user 0: 50000**  **Balance of user 1: 0**  **Job?**  **T 0 1 30000**  **Success: Transfer from user 0 to user 1 30000**  **Balance of user 0: 20000**  **Balance of user 1: 30000**  **Job?**  **Q**  **$** |