# MFC 프로그래밍

# 목차

# 피드백 리뷰

[ 추가 요구 기능 구현 ]

Ø Stored Procedure를 사용한 데이터베이스 관리

Ø 능력치를 랜덤하게 지니는 아이템 생성

❑ 일반적으로 사용되는 명사를 클래스 이름으로 사용하는 것은 좋지 않은 방법이다.

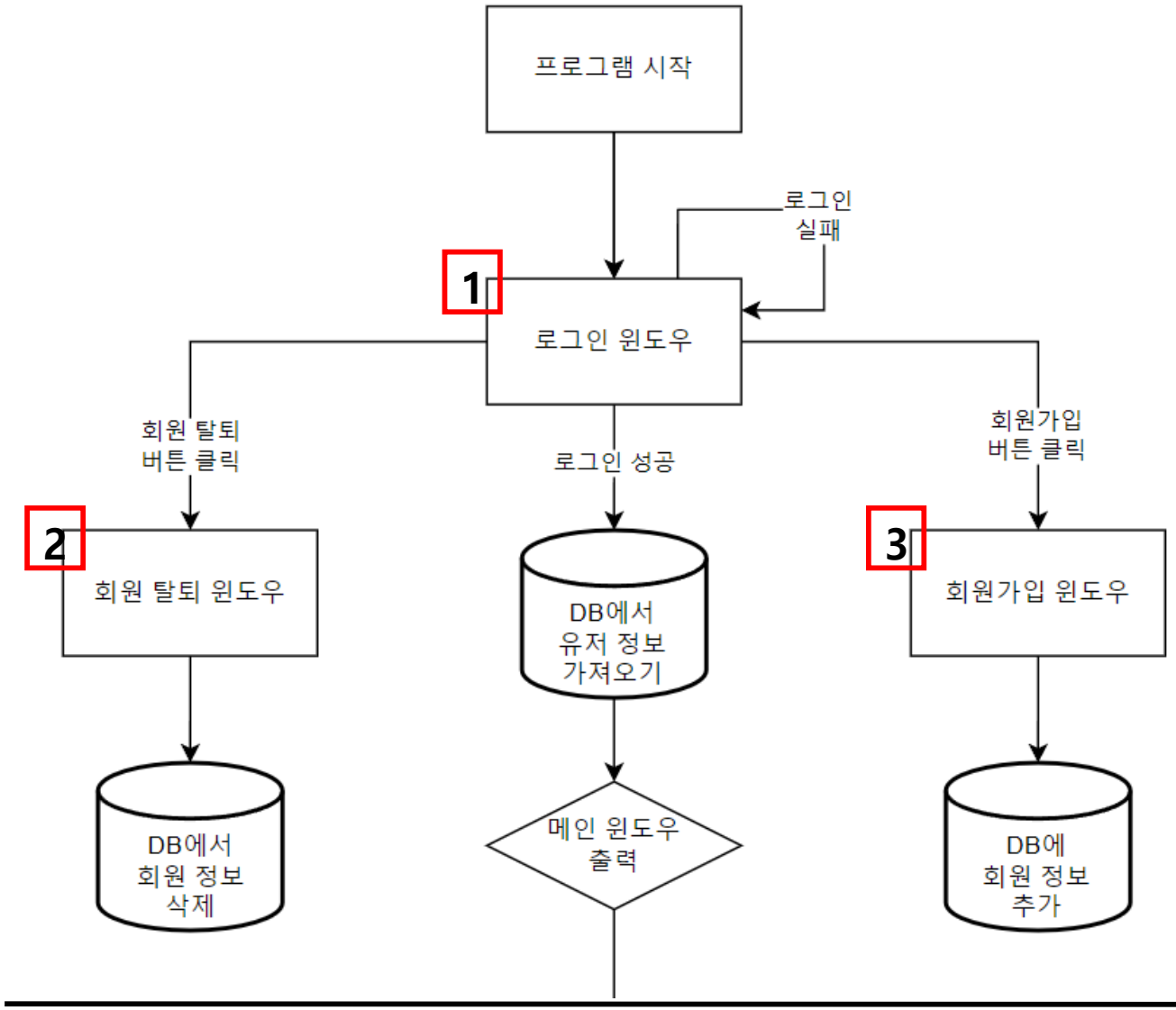
- 해당 프로젝트에서 사용한 아이템 클래스의 이름을 변경하였다.
- Ex) Item.h를 CMFCGameItem.h로 변경하였다.

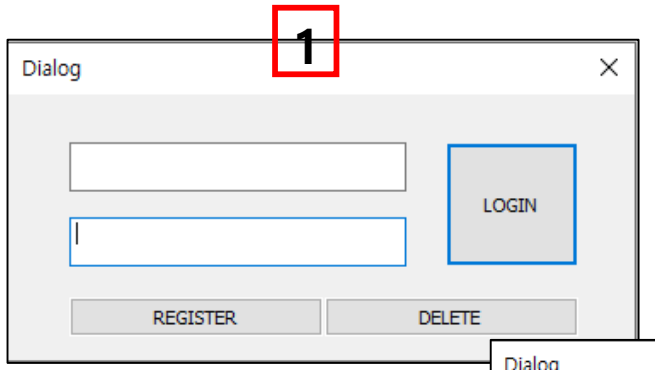
❑ 다이얼로그 클래스에 아이템 데이터를 직접적으로 저장하는 것은 좋지 않은 방법이다.

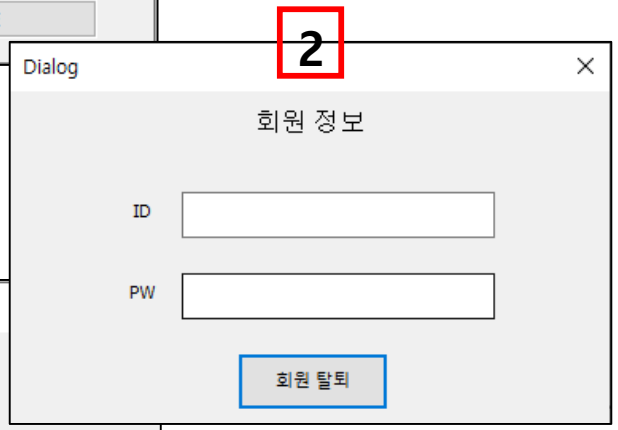
Ø 매니저 클래스들을 싱글톤 패턴으로 생성한 후 전반적인 데이터를 처리하였다.

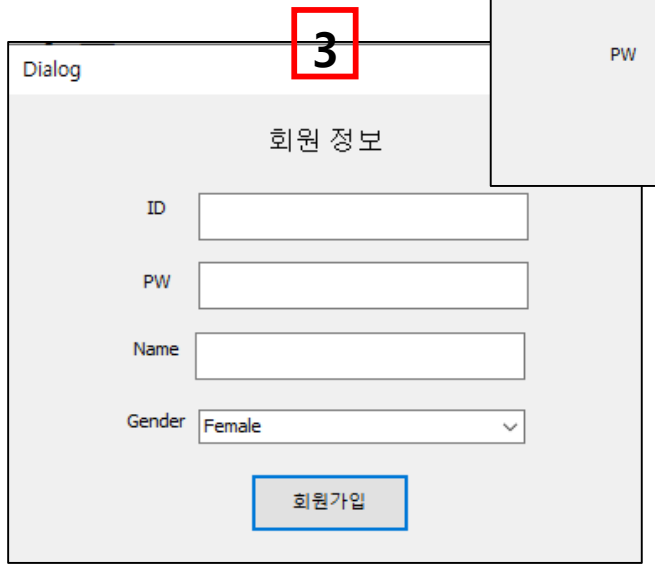
Ø Ex) 유저 스테이터스, 유저 소유 아이템, 아이템 정보 등의 데이터를 처리한다.

# 이벤트 다이어그램

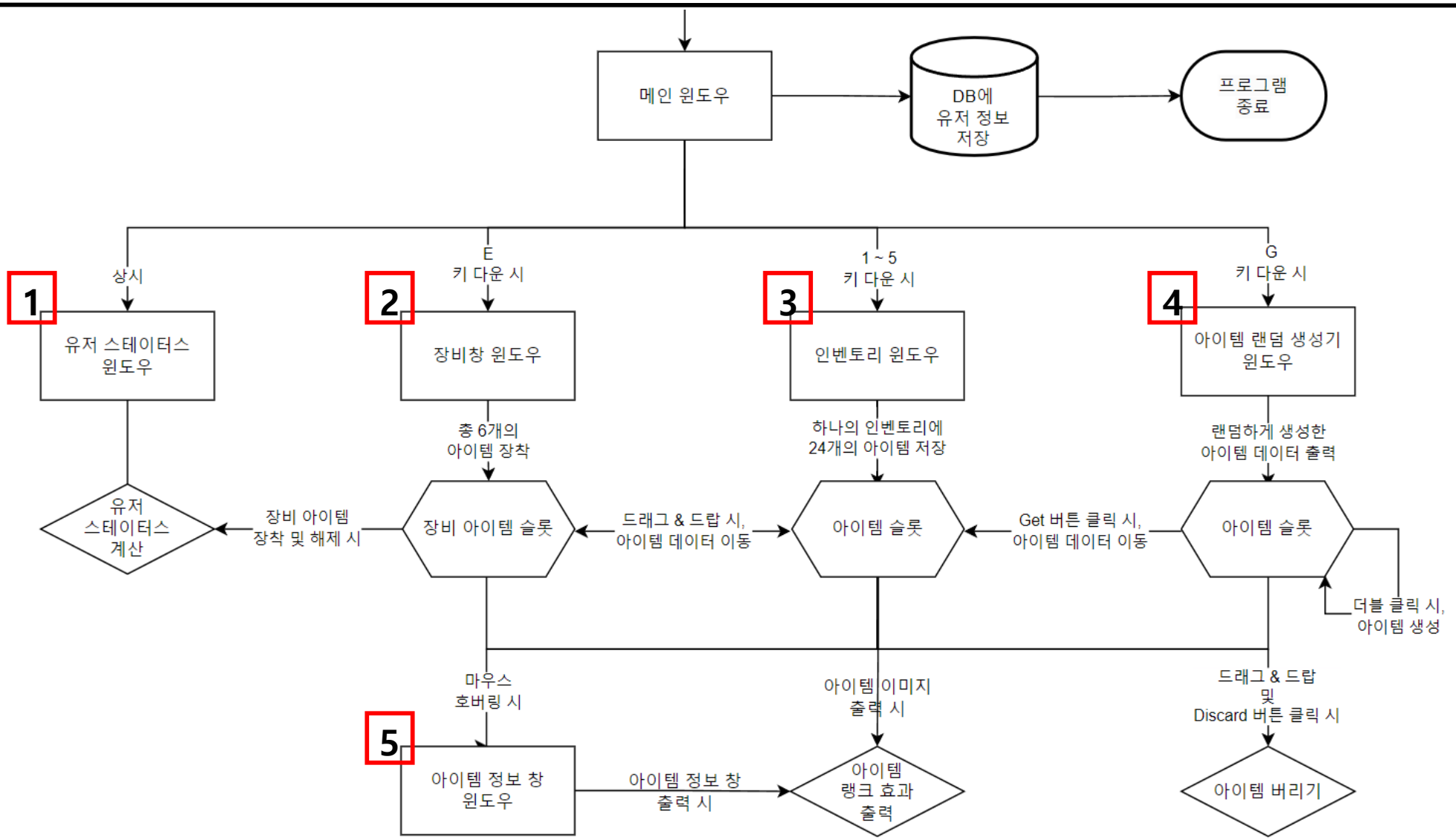

프로그램 시작
로그인
실패
1
로그인 윈도우
회원 탈퇴
버튼 클릭
로그인 성공
회원가입
버튼 클릭
2
회원 탈퇴 윈도우
DB에서
유저 정보
가져오기
3
회원가입 윈도우
DB에서
회원 정보
삭제
메인 윈도우
출력
DB에
회원 정보
추가


1
Dialog
LOGIN
REGISTER
DELETE

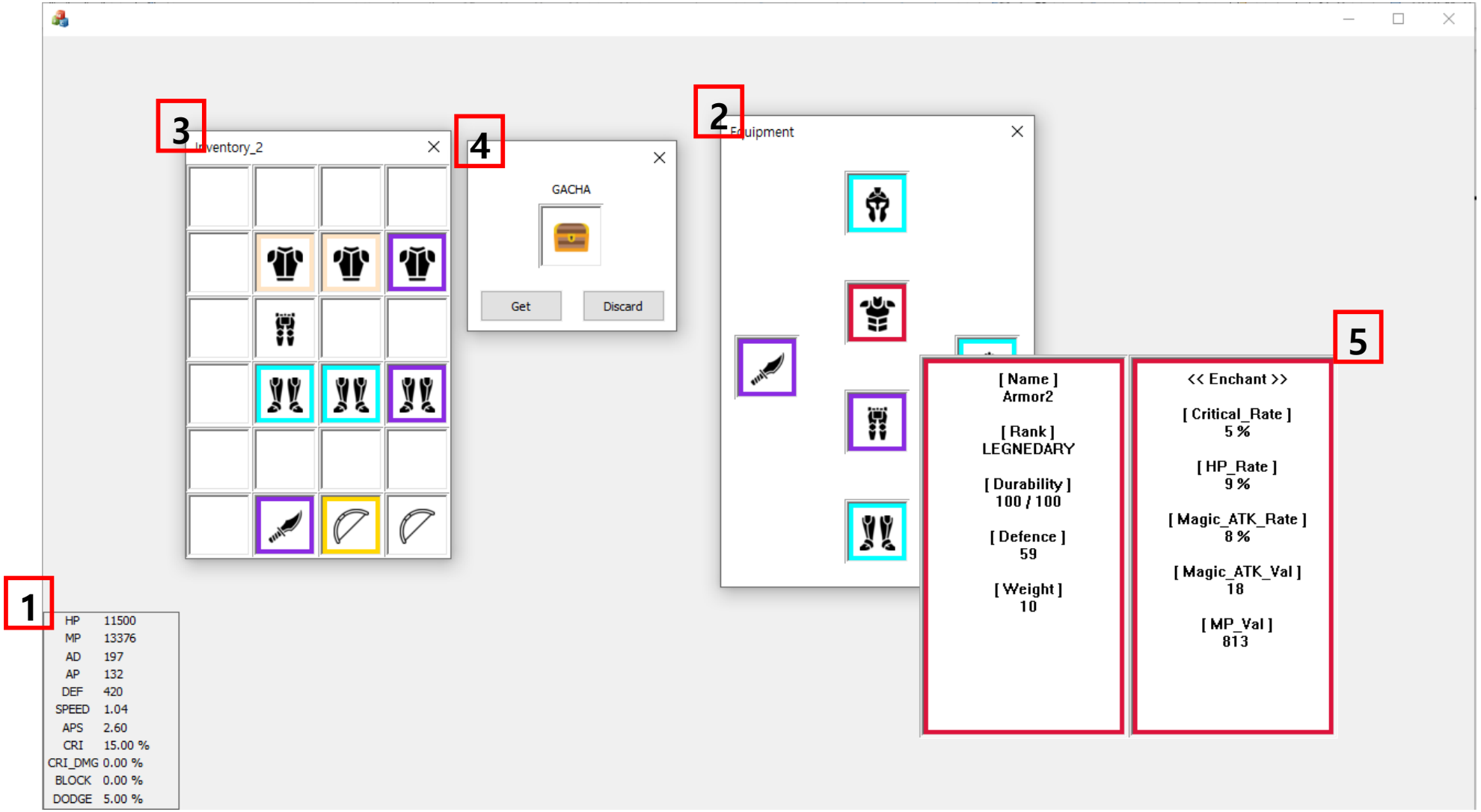

2
Dialog
회원 정보
ID
PW
회원 탈퇴


3
Dialog
회원 정보
ID
PW
Name
Gender
Female
회원가입

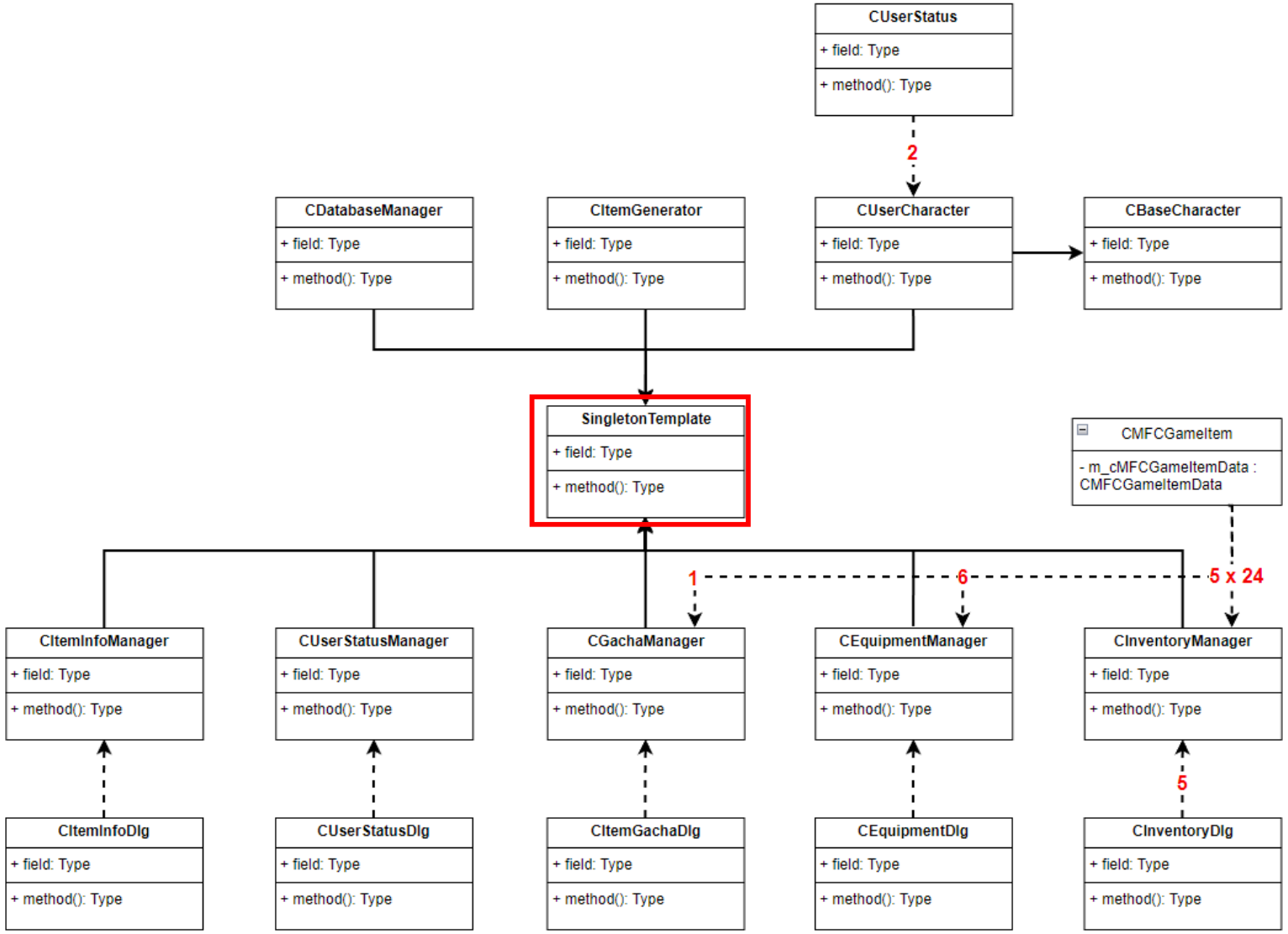
# 이벤트 다이어그램


메인 윈도우
DB에
유저 정보
저장
프로그램
종료
상시
E
키 다운 시
1 ~ 5
키 다운 시
G
키 다운 시
1
유저 스테이터스
윈도우
2
장비창 윈도우
3
인벤토리 윈도우
4
아이템 랜덤 생성기
윈도우
총 6개의
아이템 장착
하나의 인벤토리에
24개의 아이템 저장
랜덤하게 생성한
아이템 데이터 출력
유저
스테이터스
계산
장비 아이템
장착 및 해제 시
장비 아이템 슬롯
드래그 & 드랍 시,
아이템 데이터 이동
아이템 슬롯
Get 버튼 클릭 시,
아이템 데이터 이동
아이템 슬롯
더블 클릭 시,
아이템 생성
마우스
호버링 시
아이템 이미지
출력 시
드래그 & 드랍
및
Discard 버튼 클릭 시
5
아이템 정보 창
윈도우
아이템 정보 창
출력 시
아이템
랭크 효과
출력
아이템 버리기

# 이벤트 다이어그램


1
HP 11500
MP 13376
AD 197
AP 132
DEF 420
SPEED 1.04
APS 2.60
CRI 15.00 %
CRI_DMG 0.00 %
BLOCK 0.00 %
DODGE 5.00 %
2
Equipment
3
Inventory_2
4
GACHA
Get
Discard
5
[ Name ]
Armor2
[ Rank ]
LEGNEDARY
[ Durability ]
100 / 100
[ Defence ]
59
[ Weight ]
10
<< Enchant >>
[ Critical_Rate ]
5 %
[ HP_Rate ]
9 %
[ Magic_ATK_Rate ]
8 %
[ Magic_ATK_Val ]
18
[ MP_Val ]
813

# 매니저 클래스 설계

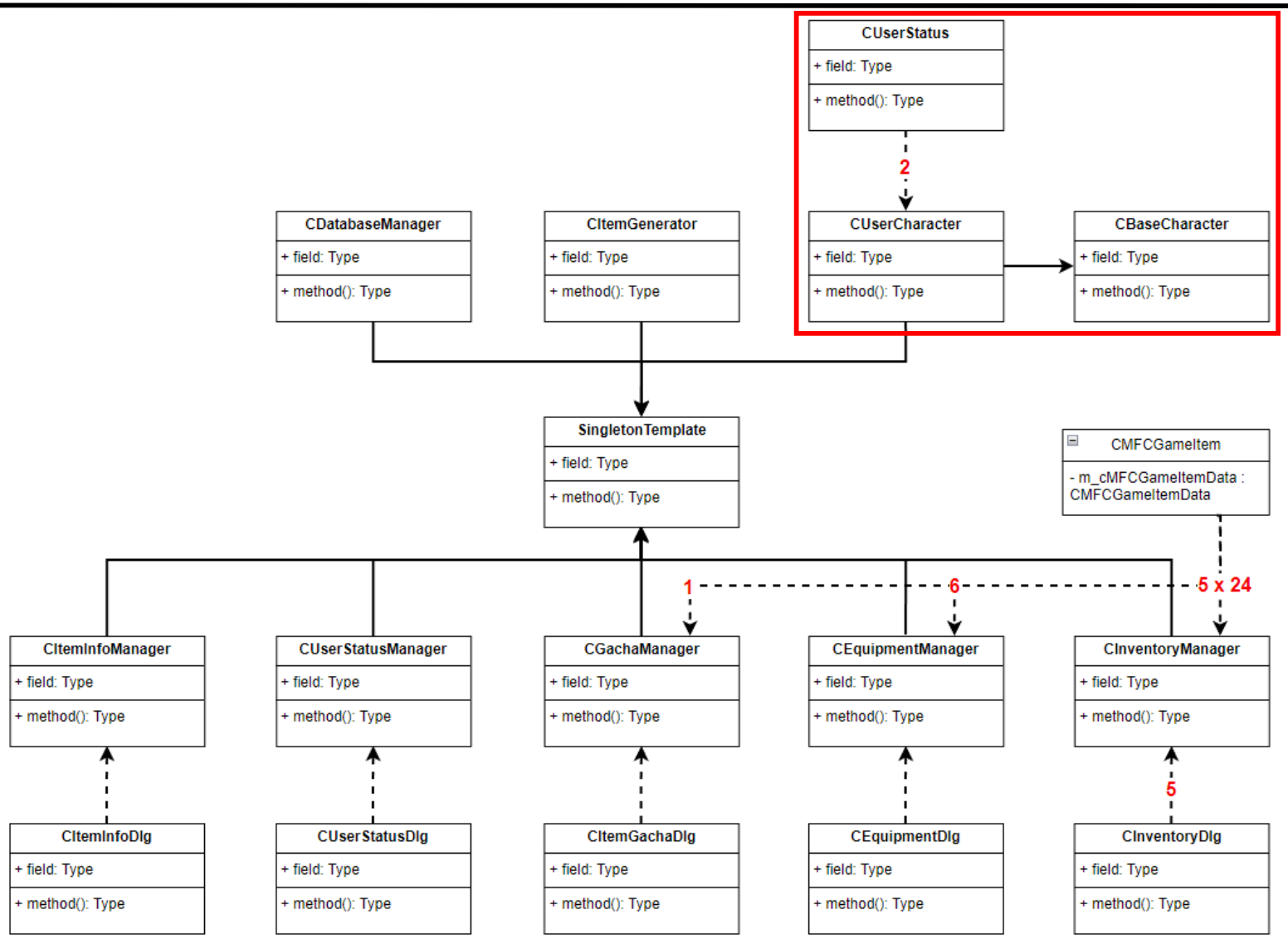



**<Singleton Template>**

- 모든 매니저 클래스는 프로그램에 하나만 존재하고, 쉽게 접근 가능하도록 하기 위해 싱글톤 클래스로 구성하였다.

- 싱글톤 패턴으로 사용할 각 클래스들은 SingleonTemplate 클래스를 상속하여 싱글톤으로 생성되게 한다.
(Local Singleton Template)

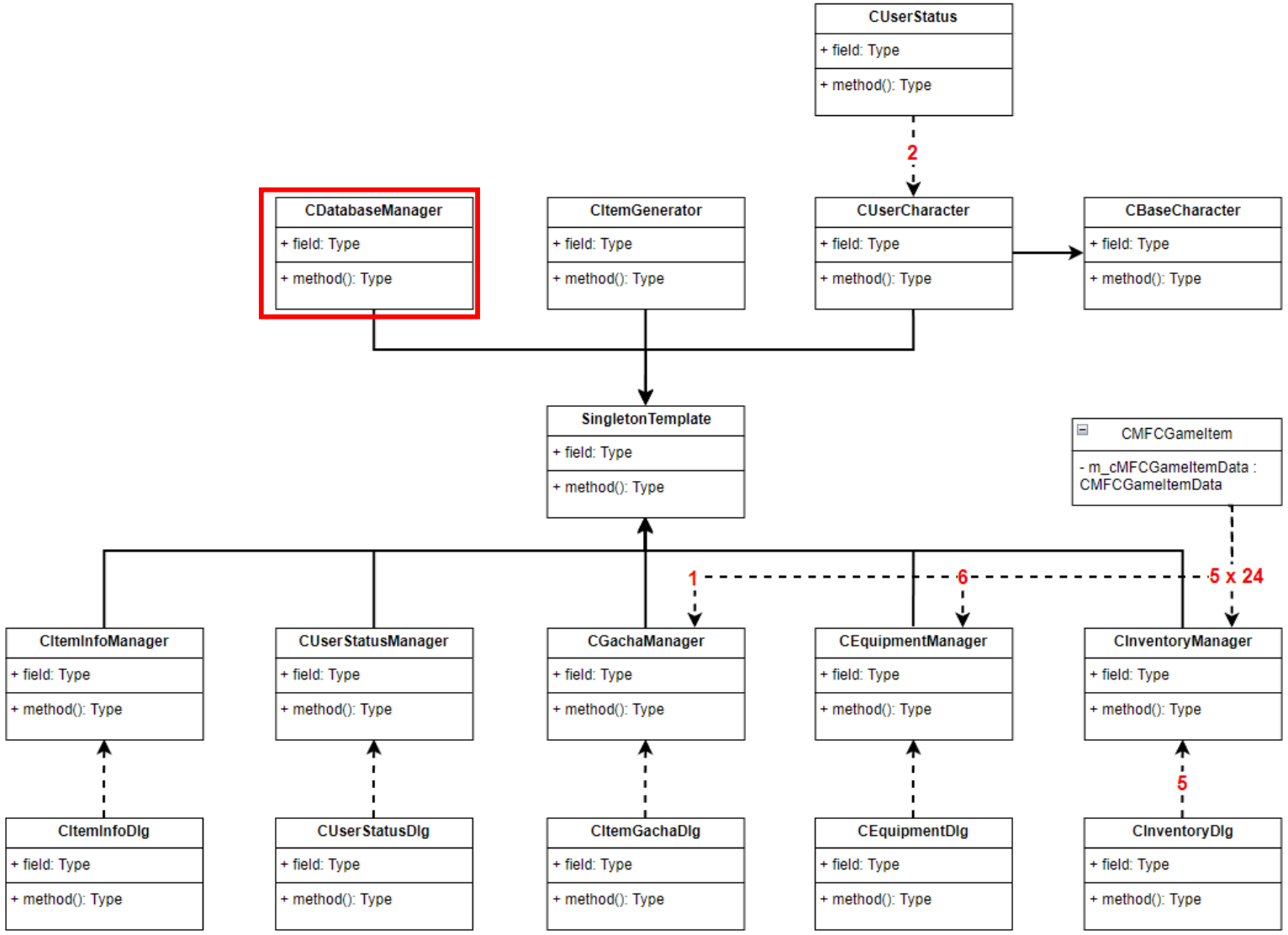
# 매니저 클래스 설계




**<CUserCharacter>**

- 캐릭터의 성별, 이름, 타입과 같은 기본 정보를 가지는 CBaseCharacter를 상속하여 유저를 지칭하는 클래스이다.
- 캐릭터의 기본 스테이터스와 아이템 장착 후의 스테이터스를 각각 구별하여 저장하기 위해 CUserStatus 클래스를 멤버 변수로 2개 생성하였다.

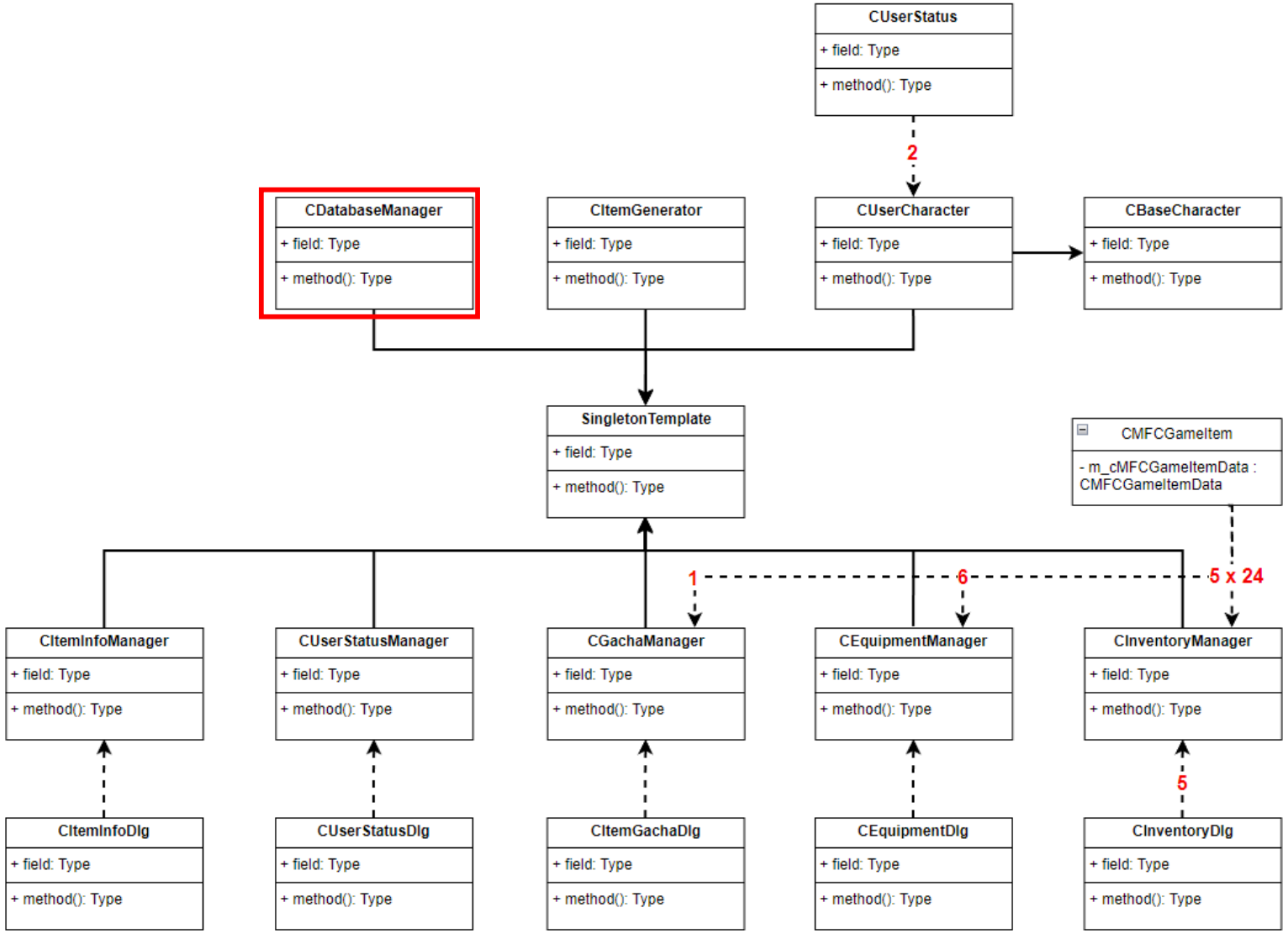
# 매니저 클래스 설계




**<CDatabaseManager>**

- DB와의 연결 후 DB에서 생성한 SP를 통해 데이터를 읽고 쓰도록 한다.
- [로그인 성공 시]
  GET_USER_INFO를 통해 유저의 스테이터스 데이터를 읽어 오고, GET_USER_ITEM을 통해 유저가 소유한 아이템 데이터들을 읽어 온다.
- [회원 가입 시]
  REGISTER을 통해 입력한 값을 토대로 유저를 DB에 추가하고, 스테이터스를 기본 값으로 설정한다.
- [회원 탈퇴 시]
  DELETE_USER를 통해 유저에 대한 모든 데이터를 삭제한다.

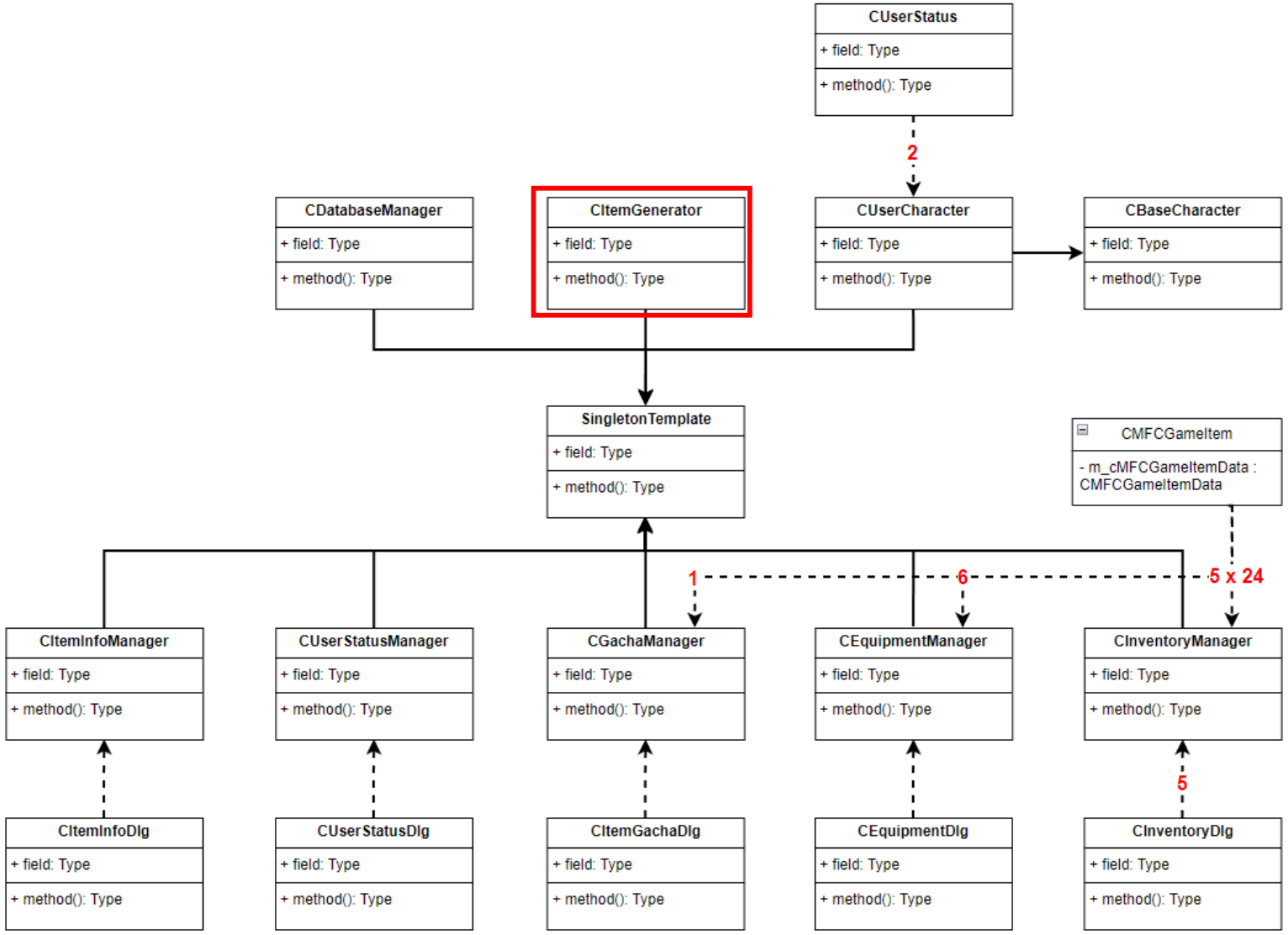
# 매니저 클래스 설계




**<CDatabaseManager>**

- DB와의 연결 후 DB에서 생성한 SP를 통해 데이터를 읽고 쓰도록 한다.
- [프로그램 종료 시]
  - INITIALIZE_USER_ITEM_DATA를 통해 유저가 소유한 아이템 데이터들을 제거한다.
  - SET_USER_INFO를 통해 유저의 스테이터스 정보를 저장한다.
  - SET_USER_ITEM_DATA를 통해 유저가 소유한 아이템 데이터를 저장한다.
  - SET_ENCHANT_DATA를 통해 유저가 소유한 아이템의 인챈트 데이터를 저장한다.
- [프로그램 종료 시] 실행되는 프로시저들은 하나의 트랜잭션으로 실행되도록 한다.

# 매니저 클래스 설계

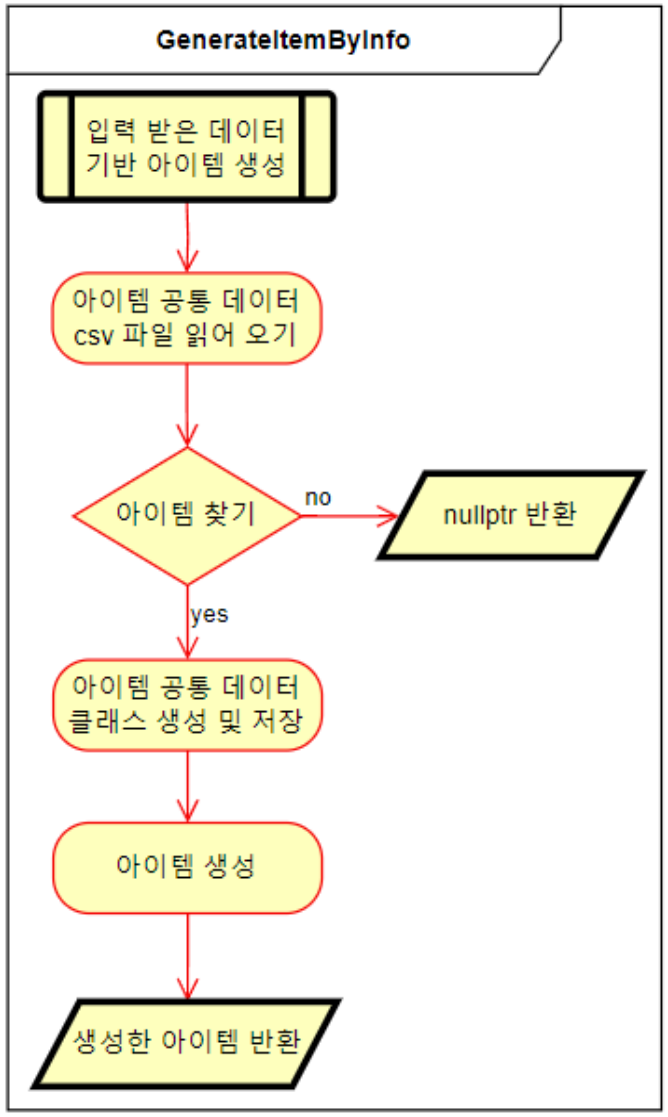



**<CItemGenerator>**

- 랜덤한 아이템을 생성하거나,
  입력 받은 아이템 데이터를 기반으로
  아이템을 생성하는 클래스이다.

- 아이템 생성 시, 아이템 공통 데이터는
  각 아이템 타입에 맞는 csv 파일을 읽어서
  생성하도록 한다.

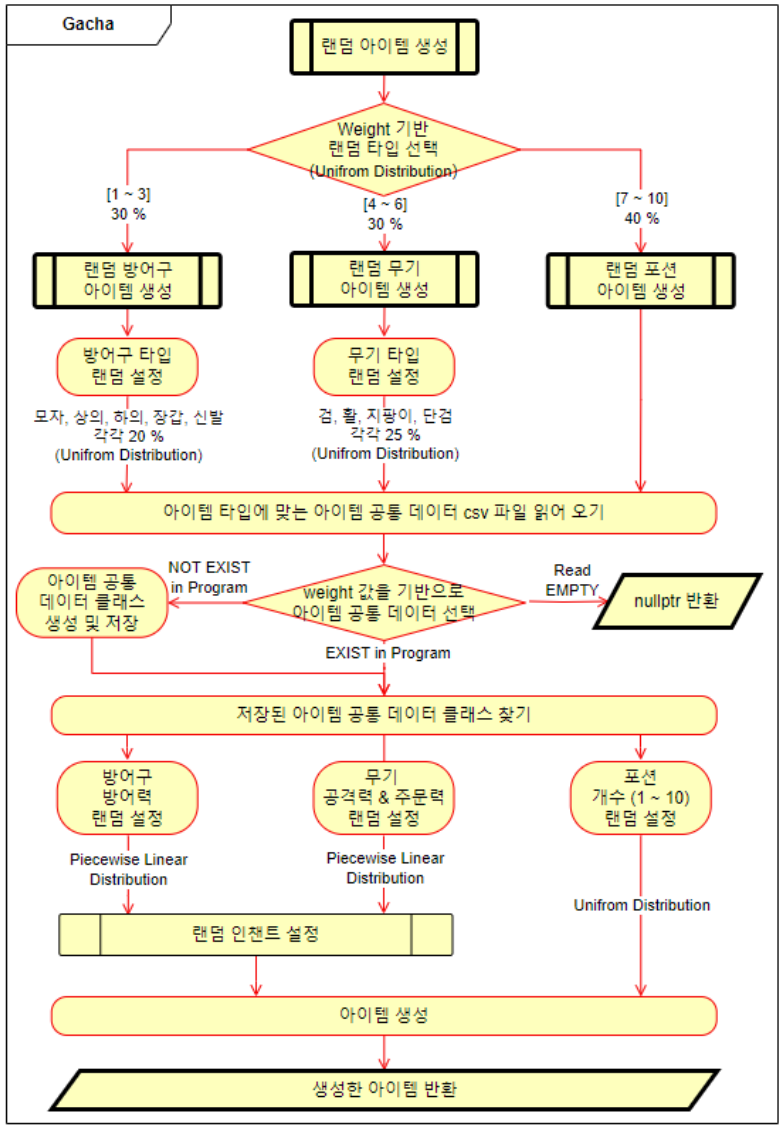
# 매니저 클래스 설계




**<CItemGenerator>**

- 랜덤한 아이템을 생성하거나, 입력 받은 아이템 데이터를 기반으로 아이템을 생성하는 클래스이다.
- 아이템 생성 시, 아이템 공통 데이터는 각 아이템 타입에 맞는 csv 파일을 읽어서 생성하도록 한다.
- [입력 받은 데이터 기반 아이템 생성 시]
  - 아이템 이름과 타입을 통해 아이템 공통 데이터를 찾아 읽어 온다.
  - 읽어 온 아이템 공통 데이터를 기반으로 아이템 공통 데이터 클래스를 생성한다.
  - 생성한 아이템 공통 데이터 클래스와 입력 받은 아이템 개별 데이터를 통해 아이템 개별 데이터 클래스를 생성한다.

# 매니저 클래스 설계



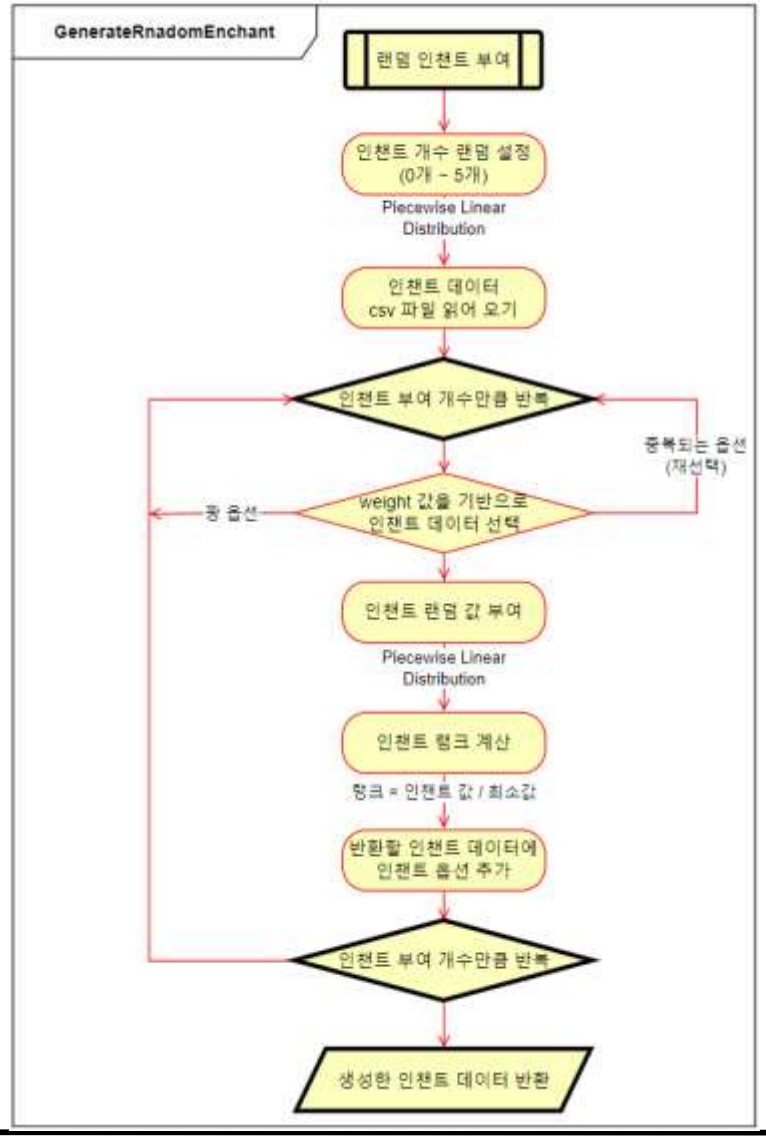

**<CItemGenerator>**

- [랜덤한 아이템 생성 시]
  - 생성할 아이템 타입을 결정한다.
  - 방어구 및 무기일 시, 세부 아이템 타입을 한 번 더 결정한다.
  - 아이템 타입에 맞는 아이템 공통 데이터 csv 파일을 읽어 온다.
  - Csv 파일에서 읽어 온 데이터들 중 weight 값을 기반으로 하나를 선택한다.
    - 선택된 아이템 공통 데이터를 통해 아이템 공통 데이터 클래스를 생성 및 저장한다. (중복되지 않는 경우)
  - 저장된 아이템 공통 데이터 클래스를 찾는다.
  - 각 아이템 타입에 맞는 개별 데이터를 랜덤으로 설정한다.
  - 랜덤하게 설정한 개별 데이터와 아이템 공통 데이터 클래스를 기반으로 개별 아이템 클래스를 생성한 후 반환한다.

- Uniform Distribution = 균등 확률 분포
  - 랜덤 타입 선택, 세부 타입 선택, 포션 개수 설정

- Piecewise Linear Distribution = 조각 별 선형 분포
  - 방어구 방어력 설정
  - 무기 공격력 & 주문력 설정

# 매니저 클래스 설계




**<CItemGenerator>**

- [랜덤한 인챈트를 장비 아이템에 부여 시]
  - 장비 아이템에 부여할 인챈트 개수를 설정한다.
  - 인챈트 데이터가 저장된 csv 파일을 읽어 온다.
  - 부여할 인챈트 개수만큼 반복
    - Weight 값을 기반으로 인챈트 데이터를 선택한다.
    - 해당 인챈트 옵션의 최소값과 최대값 사이의 랜덤한 값을 선택한다.
    - 부여된 인챈트 값에 해당 인챈트 옵션의 최소값을 나눈 값을 인챈트 랭크로 하여 계산한다.
    - 반환할 인챈트 데이터에 해당 인챈트 옵션을 추가한다.
  - 생성된 인챈트 데이터를 반환한다.

- Piecewise Linear Distribution
  - [구간] 사용자는 연속적인 구간을 정의한다.
    (최소값 ~ 중간값 ~ 최대값) (중간값 = min + (max - min) / 3)
  - [밀도 함수] 각 구간에서의 가중치를 지정하여, 특정 구간에서 더 많은 확률로 값을 생성하도록 한다.
    ( 10.0, 8.0, 4.0 )

# 매니저 클래스 설계

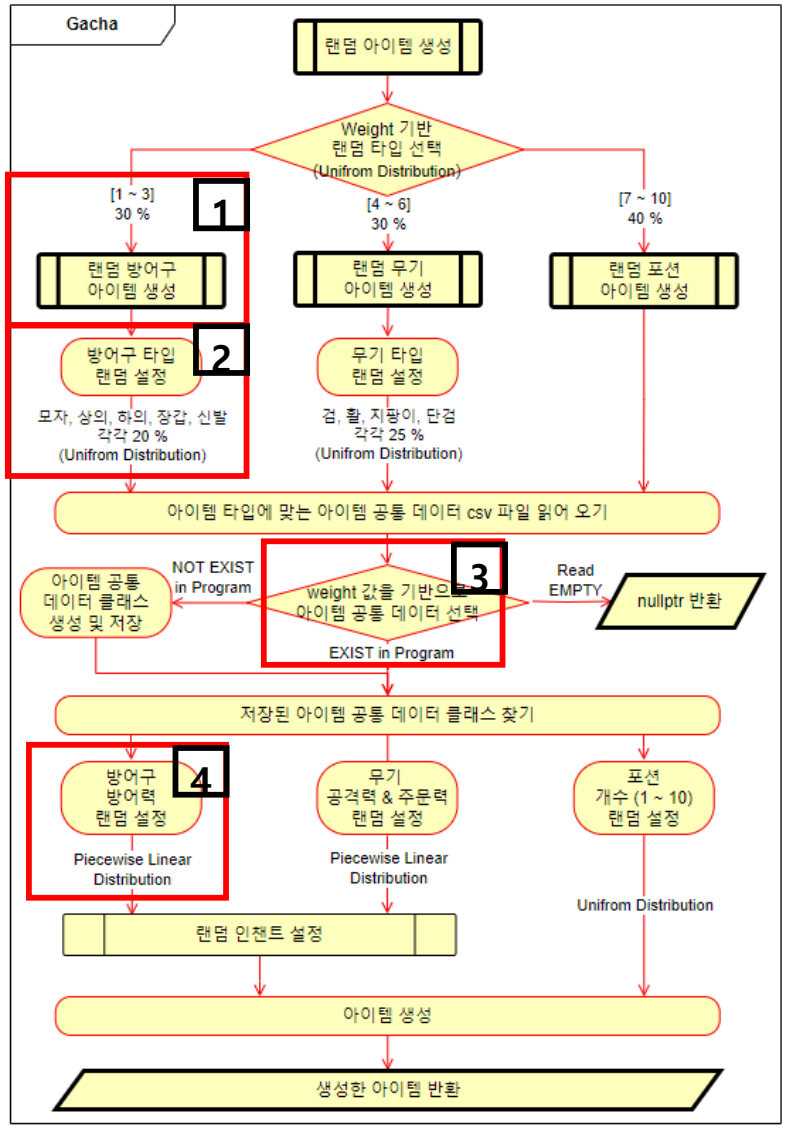
< Piecewise Linear Distribution – Example>

- 구간 = { min=1, branch=2, max=5 }
- 가중치 = { 10.0, 8.0, 4.0 }
- 두 점을 지나는 직선의 방정식
  $f(x) = w_i+(w_{i+1}-w_i)*(x-x_i)/(x_{i+1}-x_i)$
- [1, 2] -> f(x) = 10+(8-10)*(x-1)/(2-1) = 12 - 2x
- [2, 5] -> f(x) = 8+(4-8)*(x-2)/(5-2) = (32-4x) / 3
- P(x<=2) = [1,x] 사이의 f(x) 적분 값
- P(2<x<=5)
  = [1,2] 사이의 f(x) 적분 값
  + [2,x] 사이의 f(x) 적분 값

- Piecewise Linear Distribution의 결과 값은 유리수
  - 결과 값을 반올림하여 인챈트 옵션 값으로 사용
- [1.0,5.0] = P = 9 + 18 = 27
- [1.0,1.5) = P(1) = (4.75)/27 = 17.5926%
- [1.5,2.5) = P(2)
  = ([1.5,2.0)사이의 f(x) 적분 값
  + [2.0,2.5)사이의 f(x) 적분 값) / 27
  = (4.25+(3.833...))/27 = 29.938%
- [2.5,3.5) = P(3) = (3/20)/27 = 24.691%
- [3.5,4.5) = P(4) = (16/3)/27 = 19.753%
- [4.5,5.0) = P(5) = (65/30)/27 = 8.0247%

[14.5 - Piece-wise Distributions and other Examples | STAT 414 (psu.edu) 참고]

# 매니저 클래스 설계



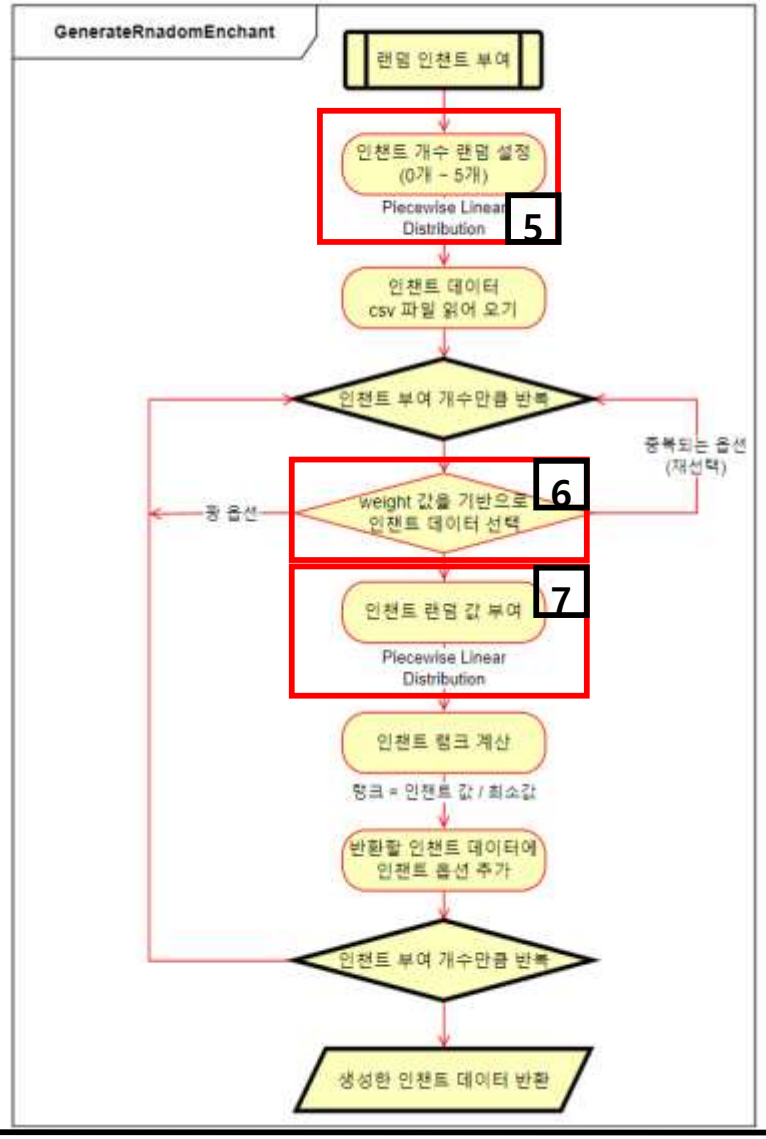

## Critical_Rate 값이 4인 인챈트 옵션을 지닌 방어력이 46이하인 Helmet1 방어구를 생성할 확률

1. 방어구 확률 = 30%
2. 방어구 중 헬멧 확률 = 20%
3. 헬멧 2가지 중 Helmet1을 고를 확률 = ½ = 50%
4. 방어구의 방어력이 53일 확률
   - f(x) = (150-2x)/7
   - P(46) = ([40.0,46.5)사이의 f(x) 적분 값 / [40,60]사이의 f(x) 적분 값) = 41.275%
5. 인챈트 부여 개수가 1개일 확률
   - f(x) = 10-x
   - P(1) = ([0.5,1.5)사이의 f(x) 적분 값 / [0,5]사이의 f(x) 적분 값) = 24%
6. 인챈트 데이터 중 Critical_Rate를 선택할 확률 = 3/93 = 3.2258%
7. Critical_Rate 옵션에 4의 값을 부여할 경우
   - f(x) = 12 - 2x
   - P(1) = ([3.5,4.5)사이의 f(x) 적분 값 / [1,5]사이의 f(x) 적분 값) = 19.753%

=> 총 확률 = 0.0002078%

- 해당 아이템 랭크 = (Critical_Rate 값 / Critical_Rate의 최소값) = 4 = Uncommon

# 매니저 클래스 설계




## Critical_Rate 값이 4인 인챈트 옵션을 지닌 방어력이 46이하인 Helmet1 방어구를 생성할 확률

1. 방어구 확률 = 30%
2. 방어구 중 헬멧 확률 = 20%
3. 헬멧 2가지 중 Helmet1을 고를 확률 = ½ = 50%
4. 방어구의 방어력이 53일 확률
   - f(x) = (150-2x)/7
   - P(46) = ([40.0,46.5)사이의 f(x) 적분 값 / [40,60]사이의 f(x) 적분 값) = 41.275%
5. 인챈트 부여 개수가 1개일 확률
   - f(x) = 10-x
   - P(1) = ([0.5,1.5)사이의 f(x) 적분 값 / [0,5]사이의 f(x) 적분 값) = 24%
6. 인챈트 데이터 중 Critical_Rate를 선택할 확률 = 3/93 = 3.2258%
7. Critical_Rate 옵션에 4의 값을 부여할 경우
   - f(x) = 12 - 2x
   - P(1) = ([3.5,4.5)사이의 f(x) 적분 값 / [1,5]사이의 f(x) 적분 값) = 19.753%

=> 총 확률 = 0.0002078%

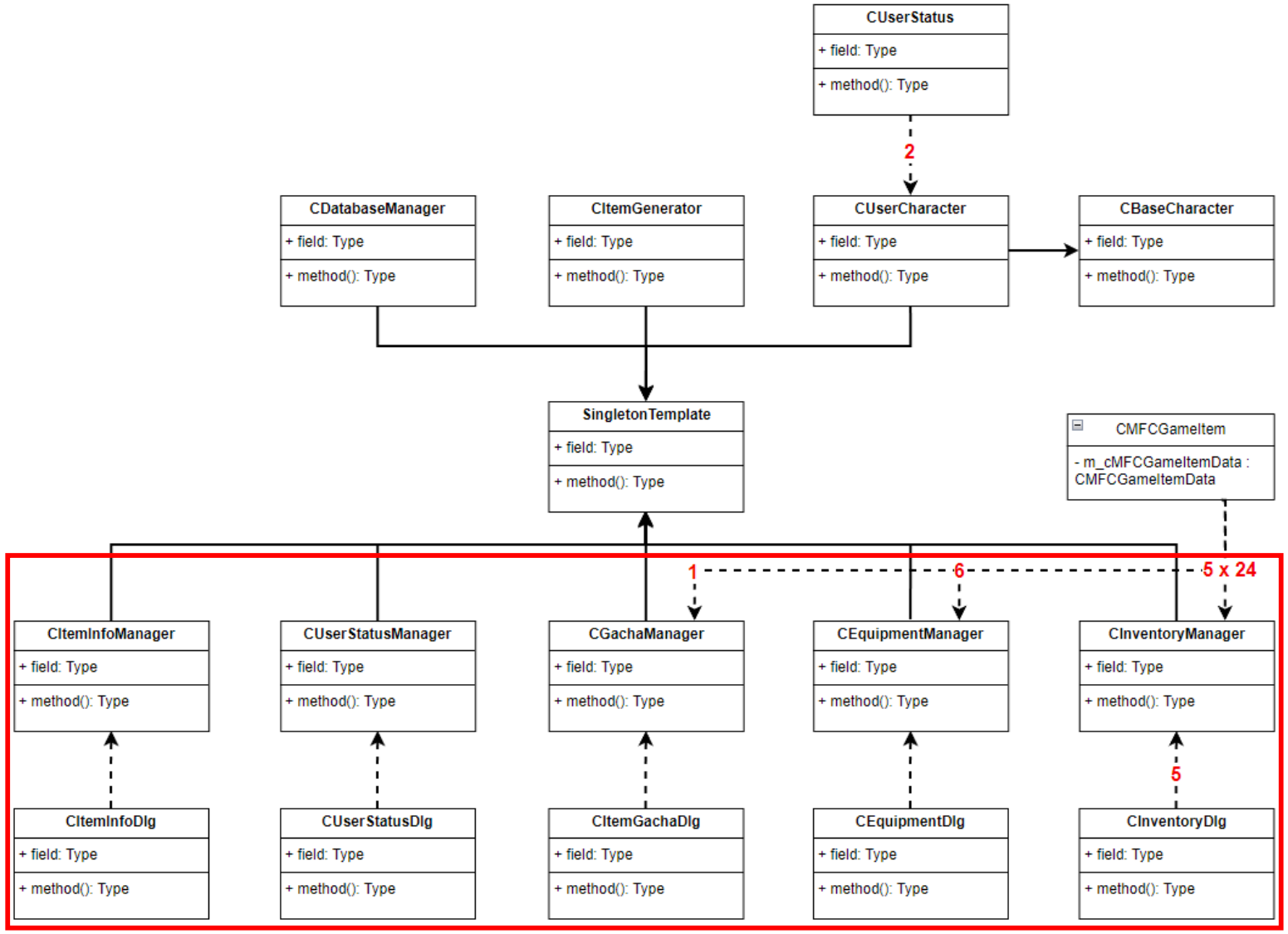
- 해당 아이템 랭크 = (Critical_Rate 값 / Critical_Rate의 최소값) = 4 = Uncommon

# 매니저 클래스 설계

```
1       #Number, ID, ArmorType, Name, FilePath, Weight, Durability, MinDefence, MaxDefence, Prob
2       1, HELMET_1, Head, Helmet1, res₩₩image₩₩helmet1.png, 10, 100, 40, 60, 1
3       2, HELMET_2, Head, Helmet2, res₩₩image₩₩helmet2.png, 10, 100, 40, 60, 1
```

```
1       #Number, Name, MinVal, MaxVal, Prob
2       1, HP_Rate, 2, 10, 5
3       2, HP_Val, 200, 1000, 7
4       3, MP_Rate, 2, 10, 5
5       4, MP_Val, 200, 1000, 7
6       5, Move_Speed_Rate, 1, 5, 2
7       6, Block_DMG_Rate, 1, 5, 1
8       7, DEF_Rate, 3, 15, 6
9       8, DEF_Val, 6, 30, 7
10      9, ATK_Rate, 2, 10, 3
11      10, ATK_Val, 5, 25, 5
12      11, Magic_ATK_Rate, 2, 10, 3
13      12, Magic_ATK_Val, 5, 25, 5
14      13, ATK_Speed_Rate, 1, 5, 2
15      14, Critical_Rate, 1, 5, 3
16      15, Critical_DMG_Rate, 1, 5, 1
17      16, Dodge_Rate, 1, 5, 1
18      17, NONE, 0, 0, 30
```

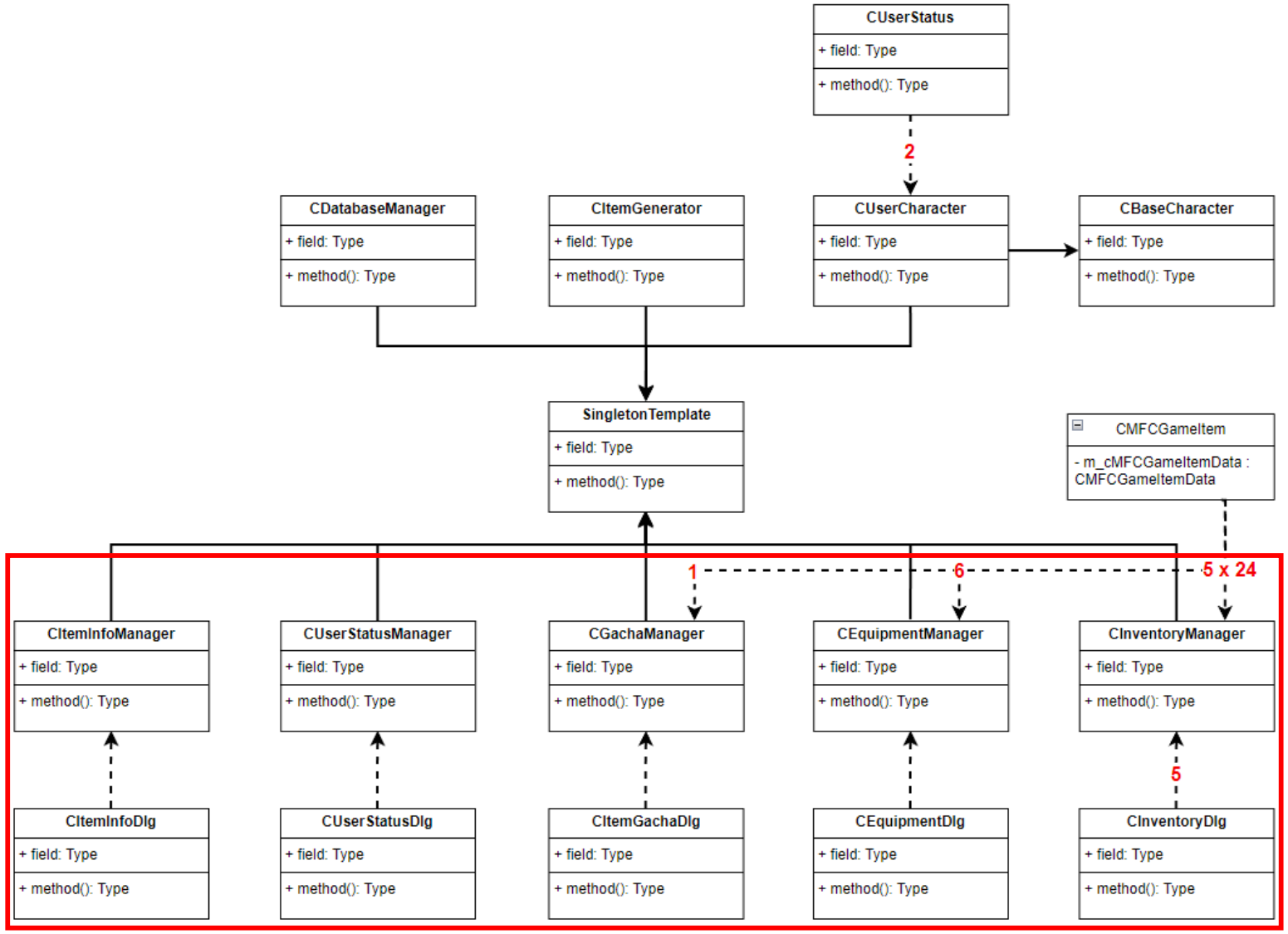
# 매니저 클래스 설계




**<Dialog Manager>**

- 해당 매니저들은 다이얼로그 클래스 대신 아이템 데이터에 직접적으로 접근하여 이벤트를 처리하기 위한 매니저 클래스이다.
- [ CItemInfoManager ]
  - 아이템 정보 출력 창을 관리한다.
  - 정보창을 Show/Hide하는 것이 주요 작업이다.
- [ CUserStatusManager ]
  - 유저의 스테이터스 출력 창을 관리한다.
  - 아이템 장착 및 해제 시, 유저 스테이터스를 업데이트하는 것이 주요 작업이다.
- [ CGachaManager ]
  - 아이템 생성 창을 관리한다.
  - 인벤토리로의 아이템 이동 등과 같이 생성된 아이템 데이터를 직접적으로 관리하는 것이 주요 작업이다.
  - 생성된 아이템을 직접적으로 소유한다.

# 매니저 클래스 설계



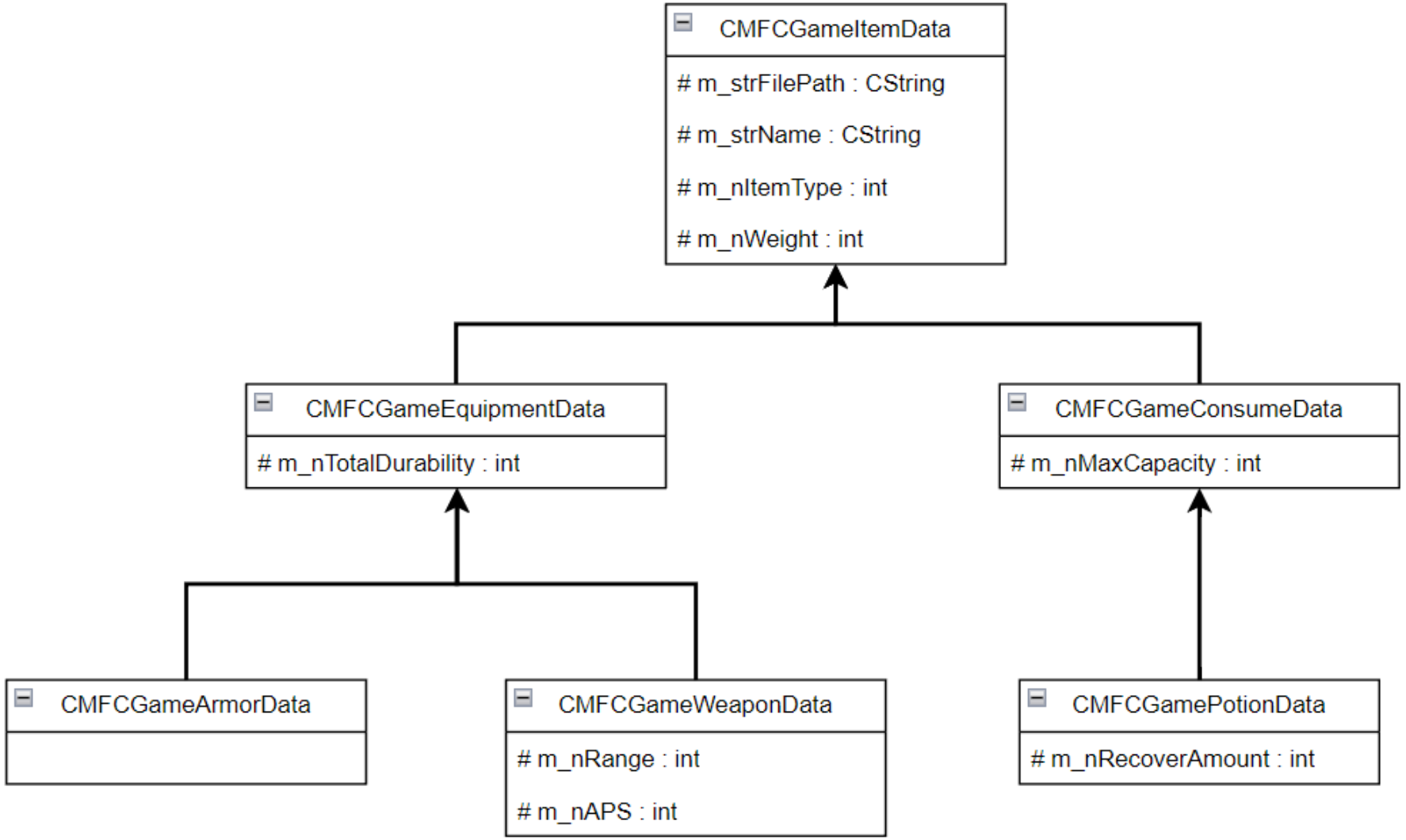

**<Dialog Manager>**

- 해당 매니저들은 다이얼로그 클래스 대신 아이템 데이터에 직접적으로 접근하여 이벤트를 처리하기 위한 매니저 클래스이다.

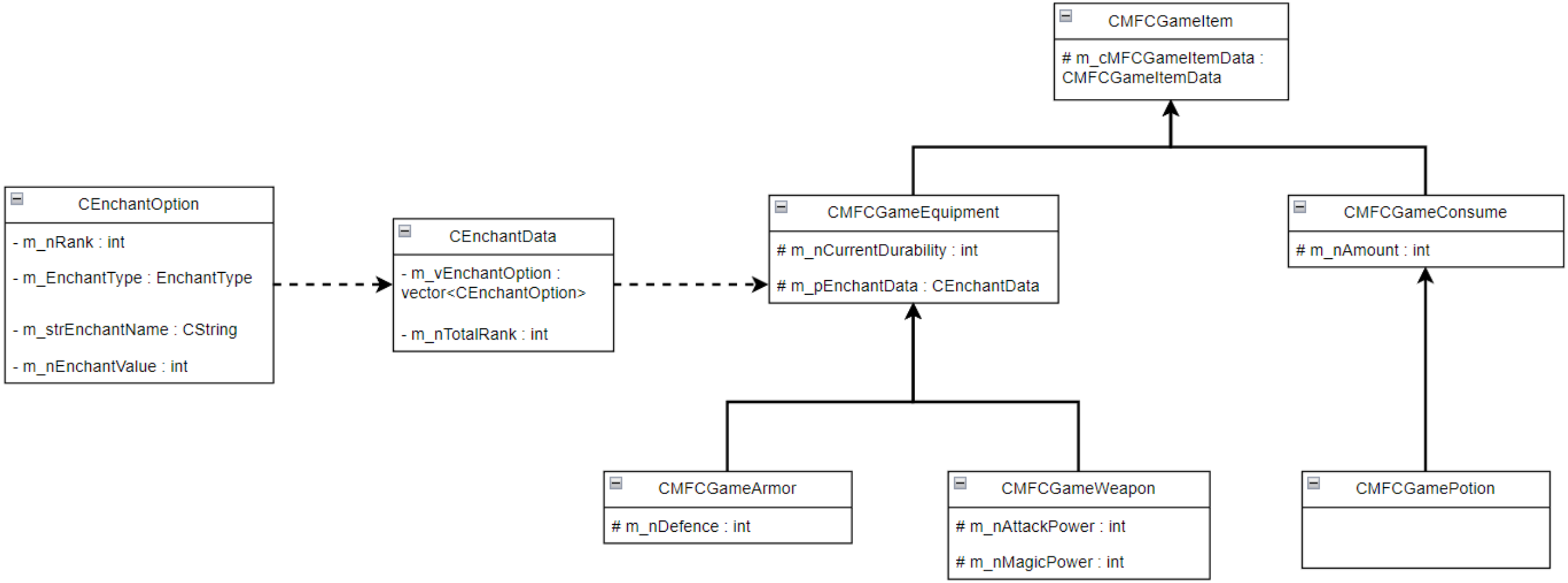
- [ CEquipmentManager ]
  - 장비창을 관리한다.
  - 장비 아이템 장착 및 해제와 같이 아이템 데이터의 직접적인 이동, 교환 및 삭제 등이 주된 작업이다.
  - 장착한 아이템을 직접적으로 소유한다.

- [ CInventoryManager ]
  - 인벤토리창 최대 5개를 관리한다.
  - 아이템 이동, 버리기, 장착 등과 같이 아이템 데이터의 직접적인 이동, 교환 및 삭제 등이 주된 작업이다.
  - 소지한 아이템을 직접적으로 소유한다.

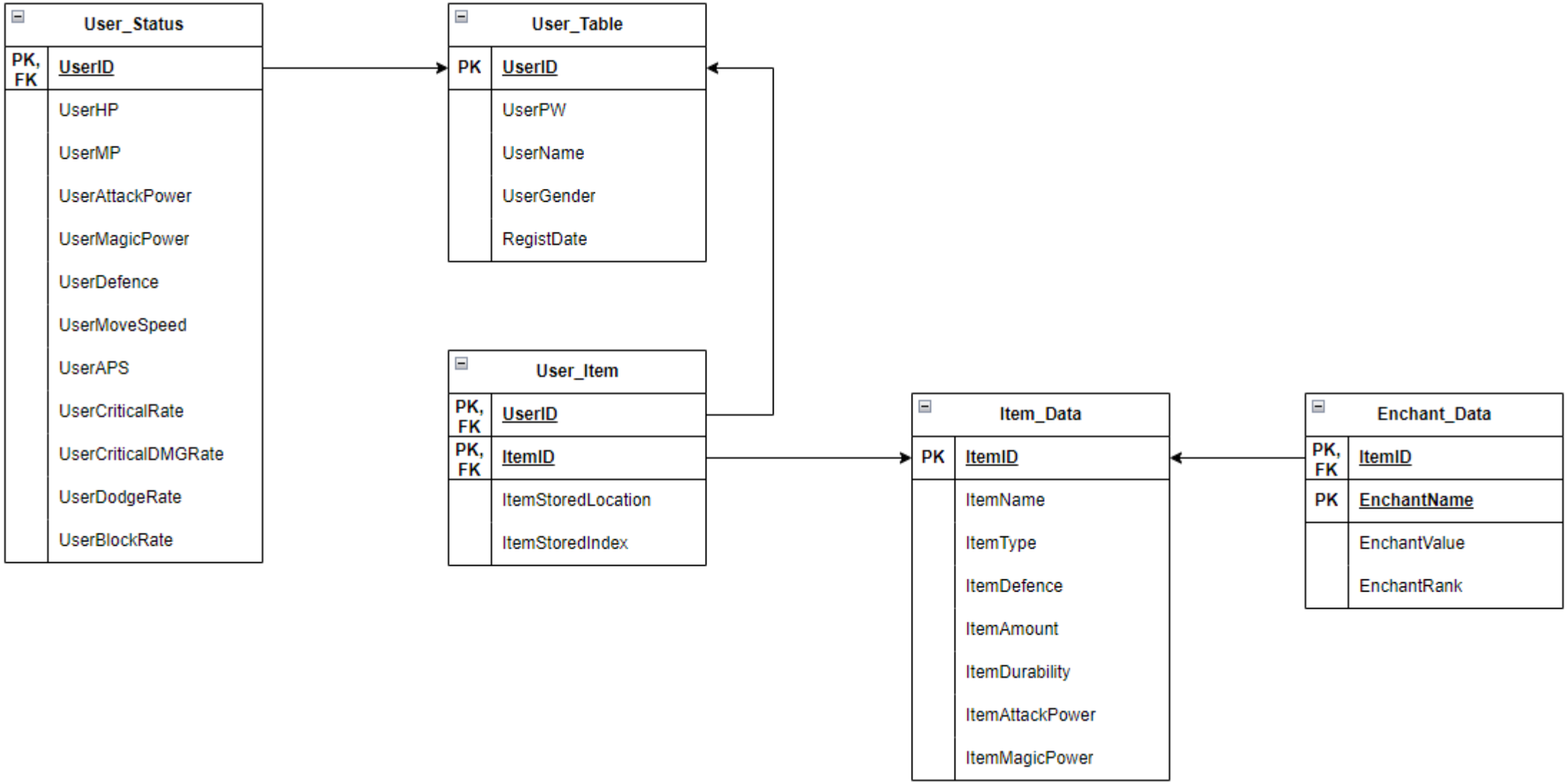
# 아이템 클래스 설계 - 공통 데이터



# 아이템 클래스 설계 - 개별 데이터



# 데이터베이스 설계


User_Status
PK, FK
UserID
UserHP
UserMP
UserAttackPower
UserMagicPower
UserDefence
UserMoveSpeed
UserAPS
UserCriticalRate
UserCriticalDMGRate
UserDodgeRate
UserBlockRate
User_Table
PK
UserID
UserPW
UserName
UserGender
RegistDate
User_Item
PK, FK
UserID
PK, FK
ItemID
ItemStoredLocation
ItemStoredIndex
Item_Data
PK
ItemID
ItemName
ItemType
ItemDefence
ItemAmount
ItemDurability
ItemAttackPower
ItemMagicPower
Enchant_Data
PK, FK
ItemID
PK
EnchantName
EnchantValue
EnchantRank

# PreCompiled Header(PCH)

- 자주 변경되지 않는 긴 소스를 미리 컴파일하여 컴파일 결과를 별도의 파일에 저장
- 다시 똑같은 헤더를 컴파일 시 해당 파일을 처음부터 컴파일하지 않고 미리 컴파일된 헤더 파일을 사용해 컴파일 속도를 월등히 향상시켜준다.

- C/C++에서 #include 메커니즘은 현재 파일에 지정된 파일의 텍스트 복사본이다. 헤더에는 다른 헤더가 포함되므로 #include를 수행하면 각 cpp 파일에 수 만 줄의 코드가 추가 될 수 있으며, 모두 매번 컴파일해야 한다.
- 이는 대규모 프로젝트에서 심각한 병목 현상이 될 수 있다.
- 미리 컴파일된 헤더는 각 헤더를 한 번씩 컴파일한 다음 컴파일된 상태를 포함된 cpp 에 포함하여 이 속도를 높인다.

- 미리 컴파일된 헤더의 장점은 거대한 시스템 라이브러리 헤더 파일 및 전혀 변경되지 않거나 드물게 변경되지 않는 기타 파일을 빌드당 한 번만 구문 분석하면 된다는 것이다.

- PCH 자체를 생성하는 데에는 특정 오버헤드가 있기 때문에 절대 변경되지 않는 항목만 PCH에 넣어야 한다.

- 다시 빌드 할 때마다 헤더 중 하나를 지속적으로 조정하여 PCH를 사용하면 다시 빌드가 느려질 수 있다.

# Char, WChar, TChar

**1. 멀티바이트 문자 집합 (MBCS) (MS 표준)**

- 아스키코드 ( 1 byte ) + 다른 문자 ( 2 byte ) 등을 포함한 문자 집합 ( 영어 & 숫자로 사용될 땐 1바이트 )
- 특정 문자 집합마다 코드 페이지가 존재한다.
- 같은 코드번호를 한글 코드 페이지로 해석하면 한글이 나오지만, 일어로 해석하면 일어가 나온다.
- 문자들이 이상하게 깨지는 문제 발생 가능성 有

**2. 유니코드 문자 집합 (WBCS) (세계 표준)**

- 아스키 코드 문자를 포함한 거의 모든 문자를 표현
- 각각의 특정 문자는 고유의 유니코드 값을 가진다.

ANSI - char은 기본적으로 1바이트이지만 한국어 등이 섞여 있으면 자동으로 2바이트로 늘어난다.

| | char | wchar | tchar | UTF-8 | UTF-16 |
|---|---|---|---|---|---|
| 문자 크기 | 1 byte | 2 byte | 컴파일 환경에 따라<br>멀티바이트 문자집합 사용시 char로<br>유니코드 문자집합 사용시 wchar로<br>처리한다. | • 영어(1Byte), 한국어 및 중국어 같은 문자(3Bytes)<br>• 유니코드 멀티바이트 인코딩 (가변 길이, **1byte ~ 4byte**) | • 대부분의 언어(2Bytes), 특수문자(4bytes)<br>• 유니코드 2바이트 인코딩 (고정 길이) |
| 타입 | ASCII, MBCS | UNICODE | | | |