Cleania's system

* **In-Unity (MonoBehavior)**
  + UI\_ItemContainer
    - 옵저버(ItemStroage 구독)
    - 프록시(ItemStorage로)
  + UI\_ItemController
    - 프록시(UI\_ItemContainer로)
  + AbilityStatus
    - 퍼사드? (옵저버!)
  + Status
    - 어댑터(StatusSO => AbilityStatus)
  + Equipable
    - 옵저버(ItemStorage 구독)
    - 프록시(ItemStorage로)
    - 어댑터(ItemInstance\_Equipment들 => AbilityStatus)
* **Out-Unity**
  + SavedData
    - 싱글톤
  + ItemStorage
    - 브릿지, 옵저버(발행자)
  + ItemInstance
    - 팩토리, 프로토타입

--------------------------------------------------------------------------------

# In-Unity

* 씬과 함께 생성
* 함께 제거
  + 게임 전체의 시스템을 맡을 수는 없음

### UI\_ItemContainer

플레이어에게 노출되는 인벤토리

* 옵저버 (ItemStorage 구독)
  + ItemStorage가 변경되면 UI\_ItemController를 생성/제거
* 프록시(ItemStorage로)
  + this에 대한 추가/삭제는 직접 실행하지 않고 구독한 ItemStorage로 전달
  + UI\_ItemController가 주로 사용

### UI\_ItemController

UI\_ItemContainer의 아이템 슬롯으로 나타남

* 프록시(UI\_ItemContainer로)
  + 유니티 이벤트에 따라 UI\_ItemContainer의 함수를 콜백

### AbilityStatus

공격받고 공격할 수 있는 객체의 스탯 수치를 산출

HP와 MP의 저장, 관리

this.AttackedBy()를 통한 데미지 계산

* 퍼사드?
  + Equipable, Status, Buffable의 값을 모아 표면적인 스탯을 저장, 반영
  + get마다 매번 계산하기보단 옵저버로 변경하여 성능 향상 가능해보임

### Status

레벨에 따른 기초 스텟 산출

subclass : Status\_ArithmeticProgress, Status\_Table

* 어댑터
  + 레벨이 오름에 따른 스텟 변화 규칙이 다르기에 적합한 방식으로 테이블을 읽어줌

### Equipable

유닛의 착용중인 장비를 저장

착용중인 장비의 스탯을 산출

* 옵저버 (ItemStorage 구독)
  + ItemStorage가 변경되면 ItemInstance\_Equipment를 가져와서 수치 반영
* 프록시 (ItemStorage로)
  + this에 대한 추가/삭제는 직접 실행하지 않고 구독한 ItemStorage로 전달
* 어댑터 (ItemInstance\_Equipment들 => AbilityStatus)
  + 여러 ItemInstance\_Equipment의 스탯 반영을 한차례 계산하여 반영할 수 있도록 산출, 저장

--------------------------------------------------------------------------------

# Out-Unity

* 어플리케이션과 함께 생성
* 어플리케이션과 함께 제거

### SavedData

게임이 시작할 때 저장된 json 파일을 파싱하여 데이터 로드

게임이 끝날 때 Serializable한 멤버들을 json으로 저장

* 싱글톤
  + ItemStorage등의 시스템 데이터를 멤버로 생성/제거 하므로 싱글톤으로 사용했었음  
    하지만 현재는 GameManager의 멤버로 넣음으로서 싱글톤 패턴을 제거
    - 게임 데이터 저장과 중앙 시스템 역할을 분리   
      => 캐릭터 선택화면에서 여러개 사용가능해짐

### ItemStorage

ItemInstance들을 저장하는 시스템 객체

subclass : ItemStorage\_LocalGrid, ItemStorage\_World, ItemStorage\_Equipment

* 브릿지
  + 모든 ItemStorage는 iItemStorage 인터페이스의   
    bool Add(ItemInstance)와 bool Remove(ItemInstance)를 구현
    - 내부에 속한 ItemInstance는 기본 추가와 기본 제거를 통해 예외 처리가 가능
* 옵저버(발행자)
  + ItemInstance의 실질적 소유자로서, 추가와 제거에 대한 갱신정보를   
    구독자들에게 발행

### ItemInstance

플레이어가 가지는 아이템 ItemInstance는 UI, 전투, 퀘스트 등 다양한 곳에 참조될 수 있지만, ItemStorage간에는 하나에만 참조되고, ItemInstance는 이를 부모로 저장

같은 아이템이어도 저장하는 데이터와 가능한 동작이 다름 => 서브클래스로 분화

subclass : ItemInstance\_Equipment, ItemInstance\_Etc

* 팩토리
  + 아이템 ID에 따라 다른 종류의 ItemInstance를 생성함
* 프로토타입
  + ItemSO를 통해 중복되는 데이터(e.g. 아이템 이름, 이미지)는 공유