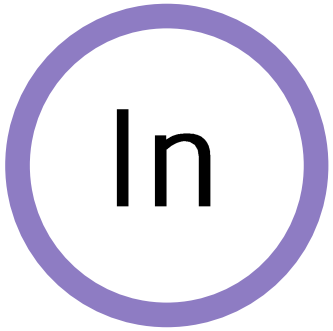
이더리움 DApp

Solidity 언어 실습



Solidity

04 Crypto Zombie(5)

Lesson1에선..

ZombieFactory 생성
- 모든 좀비를 기록하도록 state설정
- 좀비 생성 함수
- 각 좀비는 DNA를 통해 랜덤한 외모 설정

Lesson2에선..

- **게임기능**
  - 새로운 좀비 생성 기능
  - 좀비가 다른 좀비를 공격하고 먹을 수 있도록

Lesson3에선..

- 컨트랙트 생성 테크닉
- 의존성 관리하기
- 컨트랙트 보안
- 가스 최적화 및 modifier
- external view와 반복문

Crypto Zombie Lesson5

Lesson4에선..

좀비 전투시스템
- payable 제어자와 withdraw(출금)
- 난수 생성의 어려움과 외부함수접근(Oracle – 외부에서 데이터를 받아오는 안전한 방법)
- 로직구조개선
- 다양한 business logic 구현

Lesson5에선..

이더리움 상의 토큰

04 Crypto Zombie(5)

Chapter1

목표: Token?? ERC20과 ERC721
추가내용: 다음 슬라이드

아래 키워드 한번씩 들어본 적 있습니다.

- Token(토큰)??
- NFT??
- ERC20??
- ERC721

What is Token?
What is ERC?

토큰
= **스마트 컨트랙트**
= 해당 컨트랙트 안에서 누가 얼마나 많은 토큰을 가지고 있는지 기록하고
몇몇 함수로 다른 주소로 전송이 가능하게 함

이더리움에서 토큰이란?

- 몇몇 공통된 규약을 따르는 스마트 컨트랙트!
- 내부적으로 보통 다음과 같은 mapping과 함수가 존재

```
mapping(address => uint256) balances
transfer(address _to, uint256 _value)
balanceOf(address _owner)
```

- 해당 규약이 ERC20, ERC721
- 규약이란 공통된 이름의 변수와 함수!

왜 규약이 필요한가?

이를테면 모든 ERC20 토큰은 같은 이름의 함수 집합을 공유
→ 똑같은 방식으로 상호작용 가능

→ 하나의 ERC20토큰과 interaction할 수 있는 dApp을 하나 만들면,
다른 ERC20 토큰과도 interaction할 수 있음

→ 같은 함수를 호출하기 때문(to address(CA)만 변경하면 된다)
예) 암호화폐 거래소 → 하나의 전송로직만 구현해주면 모든 ERC20에 활용 가능

04 Crypto Zombie(5)

ERC 토큰

- ERC-20(https://eips.ethereum.org/EIPS/eip-20)
  ERC-20은 Ethereum Request for Comment20의 약자로 이더리움 볼록체인 네트워크에서 정한 독자적이고 대체가능한표준 토큰을 대부분의 ICO 에서 사용가능하다.
- ERC-721 (https://eips.ethereum.org/EIPS/eip-721)
  증서라고 알려진 NFT(Non-Fungible Token 대체불가능한 토큰) 표준안. (게임에서 주로 사용됨.)

  → 토큰은 대체 불가능하며 모두 제 각각의 가치(value)를 지님.

https://eips.ethereum.org/erc

표준이란 존재해야 하는 함수 인터페이스

ERC20: 화폐처럼 사용
ERC721: 대체 불가능함. 고유의 가치를 가짐. 수집품 및 digital asset에 활용

➔ 우리의 zombieContract는?? ERC20 VS ERC721

04 Crypto Zombie(5)

Chapter1 & Chapter2

목표: ERC721 – 좀비 소유권 이전하기, 다중상속
추가내용: 다음슬라이드
실습

ZombieOwnership.sol 구현 (Library를 사용하여도 되지만, 우린 학습을 위해 직접 구현!)
➔ ERC721구현을 해당 파일에서 할 예정

[요구사항]
1. ZombieBattle 상속
2. IERC721.sol 상속
   @openzeppelin/ contracts/token/ERC721/IERC721.sol

ERC721 표준 인터페이스 https://eips.ethereum.org/EIPS/eip-721

```
contract ERC721 {
  event Transfer(address indexed _from, address indexed _to, uint256 _tokenId);
  event Approval(address indexed _owner, address indexed _approved, uint256 _tokenId);

  function balanceOf(address _owner) public view returns (uint256 _balance);
  function ownerOf(uint256 _tokenId) public view returns (address _owner);
  function transfer(address _to, uint256 _tokenId) public;
  function approve(address _to, uint256 _tokenId) public;
  function takeOwnership(uint256 _tokenId) public;
}
```

링크참조:
- 현재 ERC721 인터페이스와 위 예제의 인터페이스는 조금 다름
- transfer가 이름이 **transferFrom**으로 바뀌는 등 업데이트가 존재.

@openzeppelin/ contracts/token/ERC721/ERC721.sol (구현된 컨트랙트)
@openzeppelin/ contracts/token/ERC721/IERC721.sol (인터페이스)

Solidity는 다중상속을 지원

```
contract SatoshiNakamoto is NickSzabo, HalFinney {
  // 오 이런, 이 세계의 비밀이 밝혀졌군!
}
```

- 다중상속은 여러 컨트랙트를 다중으로 상속받는것
➔ 해당 컨트랙트들의 함수, 상태변수 모두 접근 가능(private 제외)

04 Crypto Zombie(5)

ERC 721 표준 인터페이스

https://eips.ethereum.org/EIPS/eip-721

```
interface ERC721 {
    event Transfer(
        address indexed _from,
        address indexed _to,
        uint256 indexed _tokenId
    );

    event Approval(
        address indexed _owner,
        address indexed _approved,
        uint256 indexed _tokenId
    );

    event ApprovalForAll(
        address indexed _owner,
        address indexed _operator,
        bool _approved
    );

    function balanceOf(address _owner) external view returns (uint256);

    function ownerOf(uint256 _tokenId) external view returns (address);

    function safeTransferFrom(      You, 5 minutes ago • Battle 끝
        address _from,
        address _to,
        uint256 _tokenId,
        bytes data
    ) external payable;
```

```
function safeTransferFrom(
    address _from,
    address _to,
    uint256 _tokenId
) external payable;

function transferFrom(
    address _from,
    address _to,
    uint256 _tokenId
) external payable;

function approve(address _approved, uint256 _tokenId) external payable;

function setApprovalForAll(address _operator, bool _approved) external;

function getApproved(uint256 _tokenId) external view returns (address);

function isApprovedForAll(address _owner, address _operator)
    external
    view
    returns (bool);
```

04 Crypto Zombie(5)

Chapter1 & Chapter2

목표: ERC721 – 좀비 소유권 이전하기, 다중상속
추가내용: 다음슬라이드

실습

ZombieOwnership.sol 구현 (Library를 사용하여도 되지만, 우린 학습을 위해 직접 구현!)
➔ ERC721구현을 해당 파일에서 할 예정

[요구사항]
1. ZombieBattle 상속
2. IERC721.sol 상속
   @openzeppelin/ contracts/token/ERC721/IERC721.sol

ERC721 표준 인터페이스 https://eips.ethereum.org/EIPS/eip-721

```
contract ERC721 {
  event Transfer(address indexed _from, address indexed _to, uint256 _tokenId);
  event Approval(address indexed _owner, address indexed _approved, uint256 _tokenId);

  function balanceOf(address _owner) public view returns (uint256 _balance);
  function ownerOf(uint256 _tokenId) public view returns (address _owner);
  function transfer(address _to, uint256 _tokenId) public;
  function approve(address _to, uint256 _tokenId) public;
  function takeOwnership(uint256 _tokenId) public;
}
```

링크참조:
- 현재 ERC721 인터페이스와 위 예제의 인터페이스는 조금 다름
- transfer가 이름이 **transferFrom**으로 바뀌는 등 업데이트가 존재.

@openzeppelin/ contracts/token/ERC721/IERC721.sol (인터페이스) 내용

```
contract IERC721 is IERC165 {
    event Transfer(address indexed from, address indexed to, uint256 indexed tokenId);
    event Approval(address indexed owner, address indexed approved, uint256 indexed tokenId);
    event ApprovalForAll(address indexed owner, address indexed operator, bool approved);

    function balanceOf(address owner) public view returns (uint256 balance);

    function ownerOf(uint256 tokenId) public view returns (address owner);
    function safeTransferFrom(address from, address to, uint256 tokenId) public;
    function transferFrom(address from, address to, uint256 tokenId) public;
    function approve(address to, uint256 tokenId) public;
    function getApproved(uint256 tokenId) public view returns (address operator);

    function setApprovalForAll(address operator, bool _approved) public;
    function isApprovedForAll(address owner, address operator) public view returns (bool);


    function safeTransferFrom(address from, address to, uint256 tokenId, bytes memory data) public;
}
```

@openzeppelin/ contracts/token/ERC721/ERC721.sol (구현된 컨트랙트)
@openzeppelin/ contracts/token/ERC721/IERC721.sol (인터페이스)

04 Crypto Zombie(5)

Chapter3

목표: ERC721 –
자산의 양(잔고)표현, 자산의 소유주 표현(balanceOf, ownerOf)

```
contract IERC721 is IERC165 {
    event Transfer(address indexed from, address indexed to, uint256 indexed tokenId);
    event Approval(address indexed owner, address indexed approved, uint256 indexed tokenId);
    event ApprovalForAll(address indexed owner, address indexed operator, bool approved);

    function balanceOf(address owner) public view returns (uint256 balance);

    function ownerOf(uint256 tokenId) public view returns (address owner);
    function safeTransferFrom(address from, address to, uint256 tokenId) public;
    function transferFrom(address from, address to, uint256 tokenId) public;
    function approve(address to, uint256 tokenId) public;
    function getApproved(uint256 tokenId) public view returns (address operator);

    function setApprovalForAll(address operator, bool _approved) public;
    function isApprovedForAll(address owner, address operator) public view returns (bool);


    function safeTransferFrom(address from, address to, uint256 tokenId, bytes memory data) public;
}
```

실습

1. balanceOf 구현
2. ownerOf 구현

** hint: 각각 좀비의 개수와, 좀비의 주인을 반환하는 함수다.

```
function balanceOf(address owner) public view returns (uint256 balance);

function ownerOf(uint256 tokenId) public view returns (address owner);
```

balanceOf
address를 받아, 해당 address가 토큰을 얼마나 가지고 있는지 반환.
→ 우리의 경우 좀비의 개수

ownerOf
토큰ID(우리의 경우 zombieId)를 받아 소유하고 있는 사람의 address를 반환.

04 Crypto Zombie(5)

Chapter5 & Chapter 6 & Chapter 7 & Chapter 8

목표: ERC721 –
좀비 소유권 이전하기 (transferFrom, approve)

```
contract IERC721 is IERC165 {
    event Transfer(address indexed from, address indexed to, uint256 indexed tokenId);
    event Approval(address indexed owner, address indexed approved, uint256 indexed tokenId);
    event ApprovalForAll(address indexed owner, address indexed operator, bool approved);

    function balanceOf(address owner) public view returns (uint256 balance);

    function ownerOf(uint256 tokenId) public view returns (address owner);
    function safeTransferFrom(address from, address to, uint256 tokenId) public;
    function transferFrom(address from, address to, uint256 tokenId) public;
    function approve(address to, uint256 tokenId) public;
    function getApproved(uint256 tokenId) public view returns (address operator);

    function setApprovalForAll(address operator, bool _approved) public;
    function isApprovedForAll(address owner, address operator) public view returns (bool);


    function safeTransferFrom(address from, address to, uint256 tokenId, bytes memory data) public;
}
```

실습

1. Approve내용을 저장하기 위한 **mapping zombieApprovals** 만들기 (key: tokenId(zombieId), value: to)
2. transferFrom 구현 hint: transferFrom은 좀비의 원주인 혹은 approve된 계정 모두에 의해 실행 될 수 있다.
   - 확장성을 위해 _transform private 함수를 만들고 transferFrom에서 해당 함수를 호출함으로써 구현!
   - _transform에는 Transfer 라는 이벤트를 emit
3. approve 구현 (좀비의 원 주인에 의해 실행)
   - 완료시 Approval 이벤트를 emit

```
function transferFrom(address from, address to, uint256 tokenId) public;
function approve(address to, uint256 tokenId) public;
```

```
// safeTransferFrom은 최근 추가된 interface.
// transferFrom호출 전에 로직을 추가할 수 있음.(wrapper 함수라고 생각)
```

transferFrom vs approve
→ 소유권을 이전하는 방식에 차이!

transferFrom:
1. from(owner 또는 zombieOwner)으로부터 직접 호출되며 소유권을 바로 이전시킴 (msg.sender가 from)

approve:
1. from으로부터 approve가 실행(해당 token에 대해 to가 가져가도 좋아.)
2. 승인된 to로부터 transferFrom이 호출됨. (msg.sender가 to)

04 Crypto Zombie(5)

Chapter5 & Chapter 6 & Chapter 7 & Chapter 8

목표: ERC721 –
좀비 소유권 이전하기 (transferFrom, approve)

```
contract IERC721 is IERC165 {
    event Transfer(address indexed from, address indexed to, uint256 indexed tokenId);
    event Approval(address indexed owner, address indexed approved, uint256 indexed tokenId);
    event ApprovalForAll(address indexed owner, address indexed operator, bool approved);

    function balanceOf(address owner) public view returns (uint256 balance);

    function ownerOf(uint256 tokenId) public view returns (address owner);
    function safeTransferFrom(address from, address to, uint256 tokenId) public;
    function transferFrom(address from, address to, uint256 tokenId) public;
    function approve(address to, uint256 tokenId) public;
    function getApproved(uint256 tokenId) public view returns (address operator);

    function setApprovalForAll(address operator, bool _approved) public;
    function isApprovedForAll(address owner, address operator) public view returns (bool);


    function safeTransferFrom(address from, address to, uint256 tokenId, bytes memory data) public;
}
```

실습

1. Approve내용을 저장하기 위한 **mapping zombieApprovals** 만들기 (key: tokenId(zombieId), value: to)
2. transferFrom 구현 hint: transferFrom은 좀비의 원주인 혹은 approve된 계정 모두에 의해 실행 될 수 있다.
   - 확장성을 위해 _transform private 함수를 만들고 transferFrom에서 해당 함수를 호출함으로써 구현!
   - _transform에는 Transfer 라는 이벤트를 emit
3. approve 구현 (좀비의 원 주인에 의해 실행)
   - 완료시 Approval 이벤트를 emit
4. burn 함수 구현(소각) 구현
   0번 어드레스로 보내는 것!

```
function transferFrom(address from, address to, uint256 tokenId) public;
function approve(address to, uint256 tokenId) public;
```

```
// safeTransferFrom은 최근 추가된 interface.
// transferFrom호출 전에 로직을 추가할 수 있음.(wrapper 함수라고 생각)
```

transferFrom vs approve
→ 소유권을 이전하는 방식에 차이!

transferFrom:
1. from(owner 또는 zombieOwner)으로부터 직접 호출되며 소유권을 바로 이전시킴 (msg.sender가 from)

approve:
1. from으로부터 approve가 실행(해당 token에 대해 to가 가져가도 좋아.)
2. 승인된 to로부터 transferFrom이 호출됨. (msg.sender가 to)

04 Crypto Zombie(5)

Smart Contract 보안에 관하여

프로그램은 다양한 취약점이 존재한다.

1. 오버플로 언더플로 문제
2. 메시지 호출과 접근권한 제어 문제 (public vs private)
3. 리엔트런시 문제(재진입 문제) (이중거래 문제)
4. 짧은 주소 공격 (parameter 문제)
5. 잔액 조건 무효화 공격
6. 도스 공격

04 Crypto Zombie(5)

SafeMath 라이브러리

오버플로우 문제를 막기위한 라이브러리
https://docs.openzeppelin.com/contracts/4.x/

add와 sub을 조건검사를 통해 통과해야만, 실행되도록 만듬.

require: 남은 가스 되돌려줌
assert: 남은 가스 안돌려줌 (심각한 오류시 발생)

```
library SafeMath {

  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
    if (a == 0) {
      return 0;
    }
    uint256 c = a * b;
    assert(c / a == b);
    return c;
  }

  function div(uint256 a, uint256 b) internal pure returns (uint256) {
    // assert(b > 0); // Solidity automatically throws when dividing by 0
    uint256 c = a / b;
    // assert(a == b * c + a % b); // There is no case in which this doesn't hold
    return c;
  }

  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    assert(b <= a);
    return a - b;
  }

  function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    assert(c >= a);
    return c;
  }
}
```