Dapp (탈중앙화 앱)

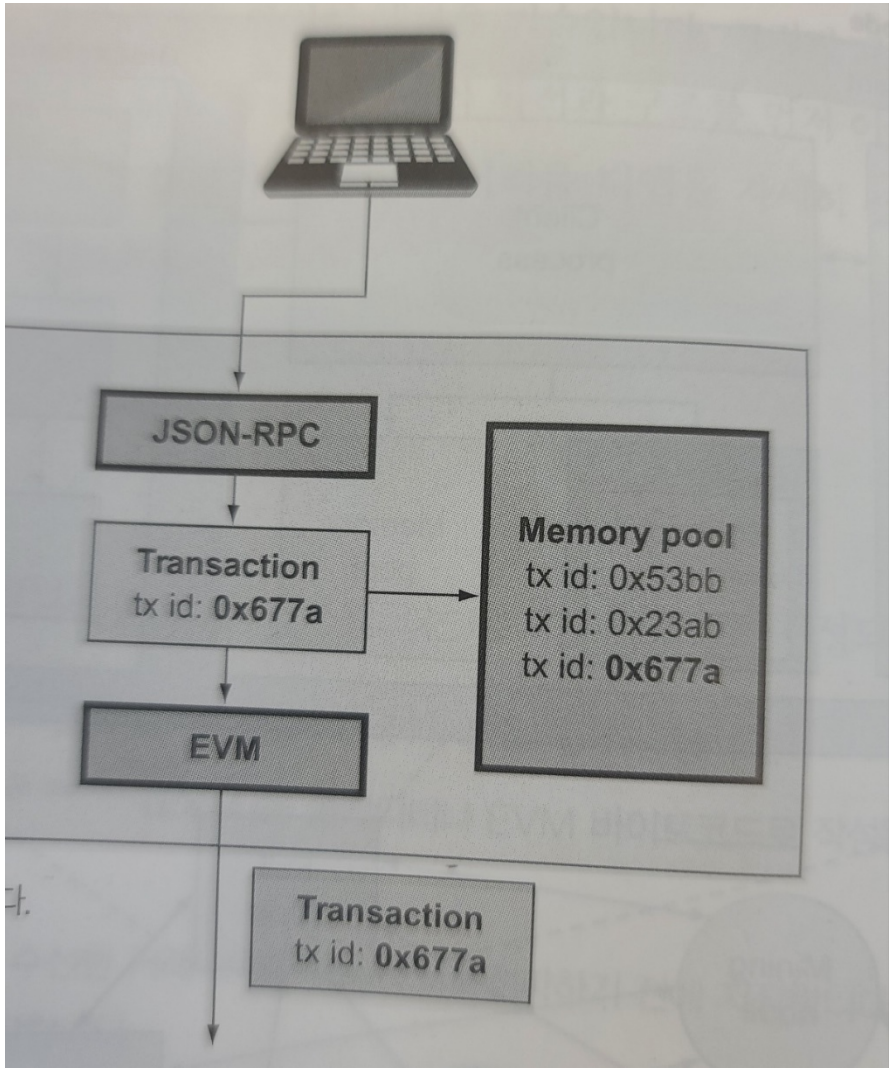
- Decentralized Application / dApp
- **클라이언트: 웹, 앱**
- **서버: P2P 블록체인 네트워크**

1) 사용자 이벤트는 웹 페이지의 js 에서 처리합니다.
2) 이벤트 발생 시 web3.js 라이브러리 함수를 사용하여 스마트 컨트랙트와 연결된 이더리움 노드와 연결하고 이벤트에 맞는 함수 A를 호출합니다.
3) 연결된 로컬 이더리움 노드 (ex. 개발자 컴퓨터) 는 메시지를 처리하고 검증한 후 피어 노드로 전달합니다.
4) 거래내역이 **채굴 노드**를 만날 때까지 전파합니다.
5) 각 노드는 새 block 을 받으면 해당 block 의 개별 거래내역이 정상적인지 + 전체 block 이 유효한지 검증합니다. 이후 block 에 존재하는 모든 거래내역을 처리하며 이 과정에서 계약 상태의 유효성을 암시적으로 확인합니다. (**합의**)
6) 검증이 성공적으로 완료되면 Confirmation 이벤트를 발생시키고, 웹 UI 를 포함하여 연결된 모든 클라이언트에 전파됩니다. 이를 UI 가 화면에 출력합니다.

EWHA-CHAIN

Ethereum Dapp

1) 사용자가 JSON-RPC 인터페이스로 스마트 컨트렉트 호출
2) 전체 노드가 생성된 거래내역 받고 메모리 풀에 저장 -> 검증을 하기 위해 거래내역이 EVM에서 실행됨
3) 검증된 거래내역은 피어노드를 통해 전파됨
4) 채굴 노드가 메모리 풀에 수신된 거래 내역 저장→ 수익성이 높은 거래내역 선택하여 EVM에서 실행 후 다음 블록에 추가 -> 메모리 풀에서 해당 내역 삭제

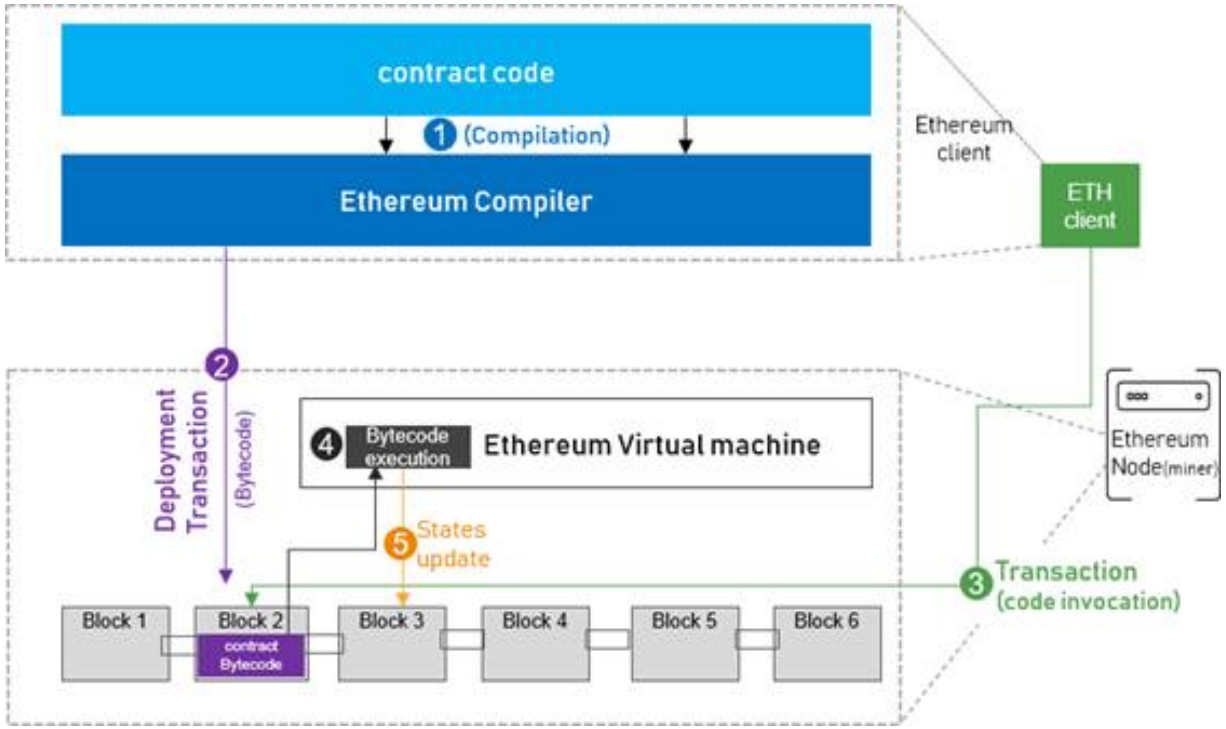


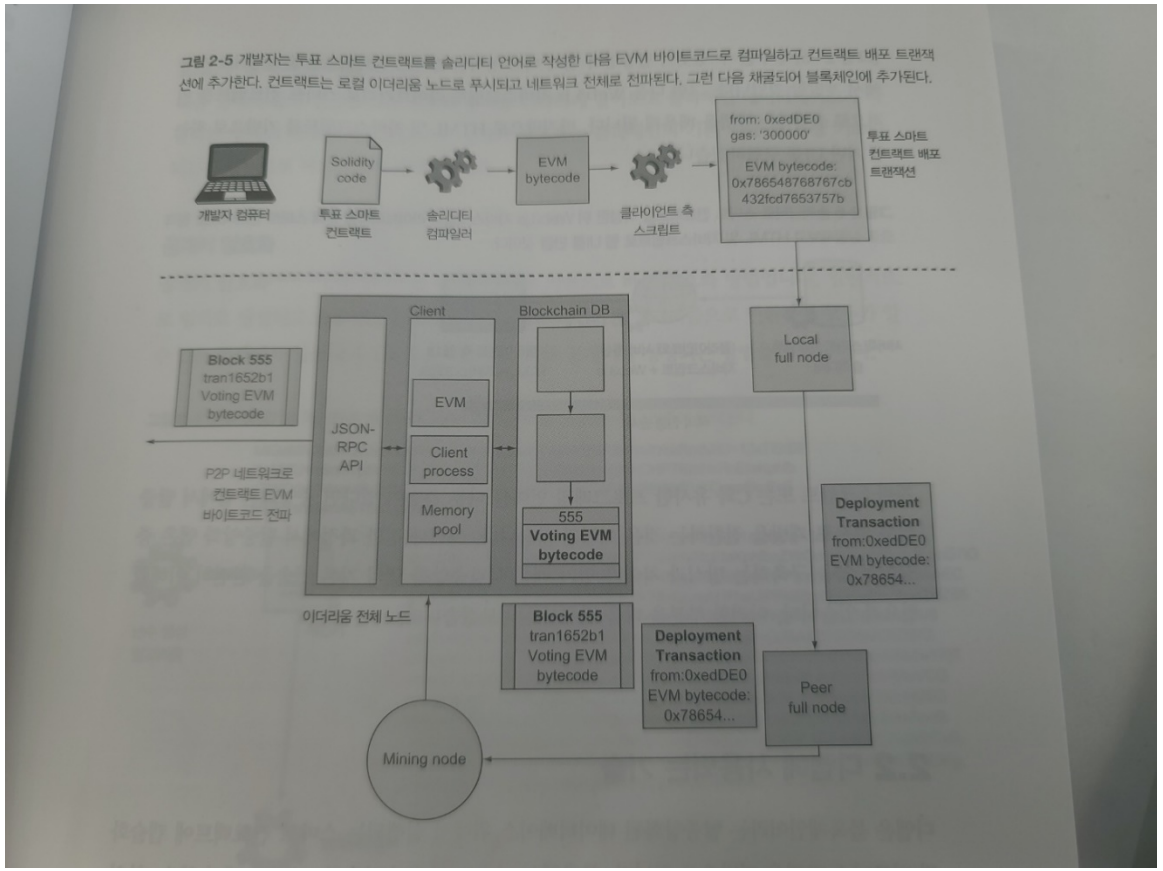
Smart Contract (스마트 컨트렉트)

- 배포 과정
  : Solidity로 작성 -> EVM 바이트 코트로 컴파일 -> 이더리움 네트워크에 배포 + 생성되는 특정 계정에 저장됨
- 사용자가 스마트 컨트렉트와 상호작용 하는 방법
  (두 가지의 메시지 유형)
  1) Call
  : 블록체인에 저장 X. 가스 소모X. -> pure, view 함수
  2) Transaction (tx)
  : 사용자 계정으로부터 tx 메시지 수신 -> EVM에서 해당 로직 실행 -> 메시지를 보낸 계정이 Ether로 수수료 지불 (가스비 -> 채굴자에게)

EWHA-CHAIN

Smart Contract (스마트 컨트렉트) - 구조도





EWHA-CHAIN

Solidity

- EVM (이더리움 가상머신)기반 스마트 컨트렉트를 작성하는 대표 언어
  - 다양한 EVM 언어 존재 (LLL, 서펜트, 바이퍼…)
  - Solidity는 이더리움 공식 Docs에서 권장, 업그레이드 및 유지보수가 잘 되어 가장 인기
- 자료 Type을 명시적으로 지정 (정적 타입)
- 객체지향 프로그래밍 지원
- IDE: Remix, VSCode 등…

EWHA-CHAIN

Solidity 문법 톺아보기

EWHA-CHAIN

pragma

```
pragma solidity ^0.8.0;
```

- pragma solididy 버전정보 형식
  -> 작성 및 컴파일 할 Solidity의 버전을 지정
  -> 새로운 컴파일러 버전에 대비

1) pragma solidity 0.8.13
-> 0.8.13 버전을 쓰겠다.

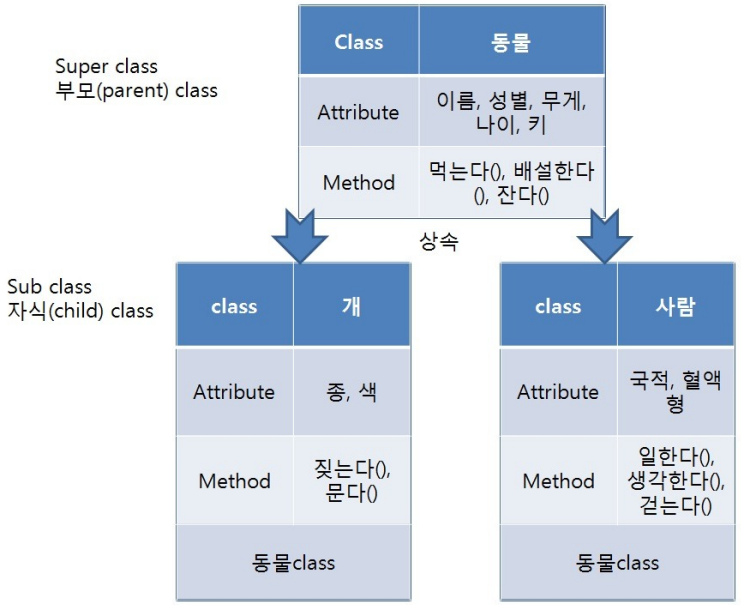
2) pragma solidity ^0.8.0
-> 0.8.0 근처의 버전을 사용하겠다

3) pragma solidity>=0.7.0 <0.9.0
-> 0.7.0 이상 0.9.0 이하 의 버전을 사용하겠다.

EWHA-CHAIN

Contract + 상속, import

- contract 컨트렉트이름 { 내용 } 형식으로 선언
- 상속:
  - 객체지향 언어에서 사용되는 개념 (상속을 받으면 자식 / 해주면 부모)
  -> 상속할 컨트렉트가 다른 파일에 있다면, import를 통해 상속할 컨트렉트 이름을 가져오기




```
import "./IERC20.sol";
import "./extensions/IERC20Metadata.sol";
import "../../utils/Context.sol";

contract ERC20 is Context, IERC20, IERC20Metadata {
```

EWHA-CHAIN

interface vs abstract contract

- interface: 틀 역할만 한다 -> 대규모 Dapp을 설계할 때 사용 (확장성)
  - 다른 곳에서 구현을 해야하므로, external로 내용 선언
  - 생성자x. 상태변수 정의 x.
  - 상속 시 인터페이스에 정의된 모든 기능을 구현해야 된다.

- abstract contract : 같은 패턴(템플릿 메소드) 주입 + 중복 제거

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
abstract contract SayHello {
    uint256 public age;
    constructor(uint256 _age ){
        age = _age;
    }

    function getAge() public virtual view returns (uint256){
        return age;
    }
    function setAge(uint256 _age) public virtual {}
    function makeMeSayHello() public  pure returns (string memory)
    {
        return "Hello";
    }
}
```

```solidity
interface ICounter {
    function count() external view returns (uint);

    function increment() external;
}
```

EWHA-CHAIN

변수 - 정수형

- int(부호 있음), uint(부호 없음, 양수만 저장 가능)으로 선언
- 8비트에서 256비트까지 8의 배수로 정확한 크기 지정 가능
  -> 작은 크기의 숫자라면, 작은 크기로 변수 선언시 Gas 줄임
- 크기를 따로 지정하지 않는다면, 자동으로 256비트로 설정

- 변수타입 변수명 = 변수값
  ex) uint256 bigNumber = 150000000000;

EWHA-CHAIN

변수 - Boolean, 문자열

- Boolean
  - 참과 거짓을 저장하는 변수
  - bool 변수명 = 값
  ex) bool isComplete = false;

- 문자열
  string name = “ewha”;

  ** 상태변수 - contract 최상단에 선언된 변수, storage
  로컬 변수 - 함수 내용 부분에 선언된 변수 (memory 기본, storage 가능)

EWHA-CHAIN

Storage vs Memory vs Calldata

- 문자열을 인자로 선언할 때 등에서 문자열 저장 영역을 선언해 줘야 함
- Storage: 상태 변수 (컨트렉트 내부 전역 변수들)처럼 블록체인에 영구적으로 유지됨
- Memory: 메모리에 임시저장
- Calldata: 함수의 매개변수값을 저장하는 영역 (메모리에 저장된 객체처럼 동작)

EWHA-CHAIN

Time Units

- uint256 형태의 timestamp 값이 기본
- 1 = 1 seconds
- Seconds, minutes, hours, days …

변수 - address

- 최대 40자리의 16진수로 구성되는 문자열 -> address
- address 변수명 = 주소
  ex) address _addr =
  0x21a31Ee1afC51d94C2eFcCAa2092aD1028285549
- 토큰, 이더 등을 전송하고 받을 수 있다
  - 컨트렉트 자체도 토큰, 이더를 가질 수 있음

배열

- 정적 배열
  - 크기를 지정해야 된다
  - 변수타입[자료개수 혹은 크기] (저장형태 - 선택) 변수
  ex) int32[5] memory fixedSlots;

- 동적 배열
  - 크기를 지정할 필요가 없다
  - int32[] unlimitedSlots;
  - push (마지막에 배열 추가), pop (마지막에 배열 삭제) 메소드 사용
  ex) unlimitedSlots.push(6);

Struct

- 다른 유형의 변수를 묶는 사용자 정의 유형
- 구조체로 배열을 만들 수도 있다
  ex) Person[] public people;
- Person.age, Person.name 으로 구조체 내부 값을 이용
  ex) Person person = new Person();
  person.age =23;

```
struct Person {
    uint age;
    string name;
}
```

EWHA-CHAIN

Function (1)

```
function eatHamburgers(string _name, uint _amout) {

}
```

- 함수 선언
  : function 함수명 (인자들 선언) {내용}
- 함수 호출
  : 함수명 (인자들에 해당되는 값들)

```
eatHamburgers("vitalik", 100);
```

- 함수 인자명은 위와 같이 언더스코어(_)로 시작해서, 함수 외부의 전역 변수와 구별하는 것이 관례
- Return 할 시, return 값을 정확히 명시

```
// returns (string)을 통해 이 함수가 문자열을 리턴함을 알 수 있습니다.
function createZombie(string _name, uint _dna) public returns (string) {
    return "create";
}
```

EWHA-CHAIN

function keywords - virtual, override

- 부모 컨트렉트의 함수에서 virtual 사용하는 함수
  -> 자식 컨트렉트에서 override를 넣고 사용
- 오버라이딩 (함수 내용 덮어씌우기) 위한 함수 키워드

contract Father

```
function getMoney() view  public virtual returns(uint256){
    return money;
}
```

```
contract Son is Father("James"){

    uint256 public earning = 0;
    function work() public {
        earning += 100;
    }

     function getMoney() view  public override returns(uint256){
        return money+earning;
    }

}
```

EWHA-CHAIN

Global namespace (전역 네임스페이스)

- Build-in-variables (솔리디티에 내장된 변수)
- block, msg, tx 등에 대한 것들 존재
- msg.sender: 함수를 호출하는 주소 (address)
  block.timestamp: 현재 시각 (uint)
  ** now : 현재 0.8.x 버전에선 block.timestamp로 대체

function keywords - 접근 제어자

- external: 외부 컨트렉트에서만 호출 (내부 X)
- public: 외부, 내부 모두 호출
- internal: 컨트렉트 내부 다른 멤버 + 상속된 컨트렉트에서만 사용
- private: 컨트렉트 내부에서만 접근 가능

```
// private 속성의 함수는 다른 컨트랙트에서 활용할 수 없습니다.
function createZombie(string _name, uint _dna) private {

}
```

```
contract Sandwich {
  uint private sandwichesEaten = 0;

  function eat() internal {
    sandwichesEaten++;
  }
}

contract BLT is Sandwich {
  uint private baconSandwichesEaten = 0;

  function eatWithBacon() public returns (string) {
    baconSandwichesEaten++;
    // eat 함수가 internal로 선언되었기 때문에 자식 컨트랙트에서 호출이 가능합니다.
    eat();
  }
}
```

EWHA-CHAIN

require vs assert vs revert

- require: 특정 조건에 부합하지 않으면 에러 발생 -> gas 환불
  - require(조건, 조건 안 맞을 시 에러메시지)
  ex. require(msg.value % 2 == 0, "Even value required.");

- assert: gas를 소비하고, 특정 조건에 부합하지 않으면 에러처리
  - assert(조건문) ex. assert(false); // 에러 발생

- revert(): 조건없이 호출 시 에러 발생 -> gas 환불
  - revert(에러메시지); 형태 ex. revert("revert");

EWHA-CHAIN

modifier (제어자)

- 다른 함수들에 대한 제어를 위한 유사 함수
- 함수 실행 전의 요구사항 충족 여부를 확인하기 위한 목적

- Openzeppelin의 onlyOwner
  : 컨트렉트의 소유자
  (컨트렉트를 배포한 사람이 기본)

```
// 사용자의 나이를 저장하기 위한 매핑
mapping (uint => uint) public age;

// 사용자가 특정 나이 이상인지 확인하는 제어자
modifier olderThan(uint _age, uint _userId) {
  require (age[_userId] >= _age);
  _;
}

// 차를 운전하기 위해서는 16살 이상이어야 하네(적어도 미국에서는).
// `olderThan` 제어자를 인수와 함께 호출하려면 이렇게 하면 되네:
function driveCar(uint _userId) public olderThan(16, _userId) {
  // 필요한 함수 내용들
}
```

_; // 조건 만족 시 호출한 함수 내용
실행할 수 있게 하는 코드

EWHA-CHAIN

단항 연산자

- + : 더하기, - : 빼기, * : 곱하기, / : 나누기, % : 나머지 연산
  ex) 5 % 2 = 1

- Solidity 0.8.x 버전 이하의 옛날 코드에서는 Over/Underflow Attack 문제로 Openzeppelin - SafeMath와 같이 자체적인 연산 함수 (내용에 오버플로우 방지 코드가 삽입됨) 사용
  https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/math/SafeMath.sol
  -> 현재 0.8.x 버전 이상에선 해당 문제 해결됨

EWHA-CHAIN

비교, 논리 연산자

- 비교 연산자

==	두 값이 같은지 여부를 확인하고, 같으면 true를 반환하고 그렇지 않으면 false를 반환
!=	두 값이 같은지 여부를 확인하고 같지 않으면 true를 반환하며 그렇지 않으면 true반환
>	왼쪽 값이 오른쪽보다 큰지 여부를 확인하고 크면 true를 반환하고 그렇지 않으면 false
<	왼쪽 값이 오른쪽보다 작은지 확인하고, 작으면 true를 반환하고 그렇지 않으면 false
>=	왼쪽 값이 오른쪽보다 크고 같은지 여부를 확인하고, 크거나 같으면 true를 반환하고 그렇지 않으면 false 반환
<=	왼쪽 값이 오른쪽보다 작은지 확인하고, 작거나 같으면 true를 반환하고 그렇지 않으면 false 반환

- 논리 연산자

&&	두 조건이 모두 참이면 true를 반환하고 하나 또는 둘 다 거짓이면 false를 반환
||	하나 또는 두 조건 모두 참이면 true를 반환하고 둘 다 거짓이면 false를 반환
!	true를 false로 false 를 true로 반환

EWHA-CHAIN

조건문 - If, else if, else

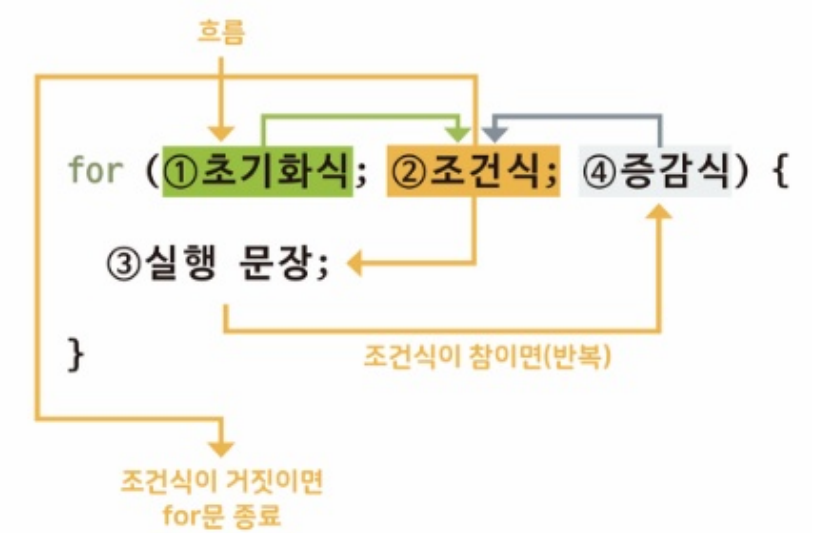
- If(1번조건) {내용}
  else if (1번 조건이 아닌 범위에서 2번 조건) {내용}
  else (1,2번 조건이 아닌 범위에서 3번 조건) {내용}
- 다른 언어와 유사

```
if (expression 1) {
   Statement(s) to be executed if expression 1 is true
} else if (expression 2) {
   Statement(s) to be executed if expression 2 is true
} else if (expression 3) {
   Statement(s) to be executed if expression 3 is true
} else {
   Statement(s) to be executed if no expression is true
}
```

```
    if( a > b && a > c) {   // if else statement
       result = a;
    } else if( b > a && b > c ){
       result = b;
    } else {
       result = c;
    }
    return integerToString(result);
}
```

EWHA-CHAIN

반복문 - for



- break: 반복문 탈출시 호출
- 이 외에도 while, do-while 등 존재
  -> 다른 언어와 유사

```
pragma solidity ^0.4.19;


contract ForLoopExample {

    mapping (uint => uint) blockNumber;
    uint counter;

    event uintNumber(uint);

    function SetNumber()  {

        blockNumber[counter++] = block.number;

    }

    function getNumbers() {

        for (uint i=0; i < counter; i++){
            uintNumber( blockNumber[i]  );
        }

    }
}
```

EWHA-CHAIN

Mapping

- Key - Value 구조로 데이터를 저장
- Mapping 값에 접근하려면
  (매핑이름)[키] 로 접근

```
// key: uint 형, value: address 형
mapping(uint => address) public zombieToOwner;

// key: address 형, value: uint 형
mapping(address => uint) ownerZombieCount;
```

```
mapping(uint => address) public zombieToOwner;

// uint형 키 0에 호출한 사람의 주소(address)가 할당된 모습입니다.
zombieToOwner[0] = msg.sender;
```

EWHA-CHAIN

View vs Pure

- 둘 다 가스를 소모하지 않음
- View : 컨트렉트 내부 변수를 반환만 한다. 값 변경X
- Pure : 컨트렉트 내부 접근 X. 해당 함수 내부에서만 동작

```
contract Example {
    uint256 public a = 5

    function exampleA() view public returns (uint256) {
          return a;
    }

    function exampleB() pure public returns (uint256) {
    uint256 public b = 3
        return b;
    }
}
```

** 상태를 변경할 때
- 상태 변수에 쓰기
- 이벤트 발생
- 컨트렉트 생성 혹은 파기
- 이더 전송
- view 혹은 pure이 아닐 때
- 저수준 (call() 혹은 특정 인라인 어셈블리 옵코드 사용)

EWHA-CHAIN

Constructor

- 컨트렉트가 생성될 때 딱 한 번! 실행되는 함수!!
- 함수를 배포할 때 넣어줘야 되는 값들 , 혹은 실행해야 되는 함수들….

```
// SPDX-License-Identifier:GPL-30
pragma solidity >= 0.7.0 < 0.9.0;

contract A{

    string public name;
    uint256 public age;

    constructor(string memory _name, uint256 _age){
        name = _name;
        age = _age;
    }

}

contract B{

  A instance = new A("Alice", 52);

}
```

EWHA-CHAIN

이벤트

```
// SPDX-License-Identifier: MIT

pragma solidity >=0.8.0 <0.9.0;

contract lec13 {

    event info(string name, uint256 money);

    function sendMoney() public {
        emit info("KimDaeJin", 1000);
    }
}
```

- 이벤트 정의
  event 이벤트명 (파라미터)
- 이벤트 방출 - 정의에 따름
  emit 이벤트명 (위에 정의된 파라미터)
- 이벤트가 필요한 이유: 프론트엔드에서의 처리를 위해 (주요 이유)
  + tx 로깅을 통한 분석에 활용

EWHA-CHAIN

기타 솔리디티 개념들

- payable 키워드: 함수에서 Ether을 지불 가능하도록 명시
  ex) function getSTockPRice(string_stockTicker) payable returns~
- keccak : 난수 생성 함수

```
keccak256("aaaab");
//6e91ec6b618bb462a4a6ee5aa2cb0e9cf30f7a052bb467b0ba58b8748c00d2e5
```

- Fallback 함수 (익명의 payable 함수)
  - 1번 선언 가능, 다른 함수를 통해 호출하여 가스 예산 최소화
  ex) function() payable{}

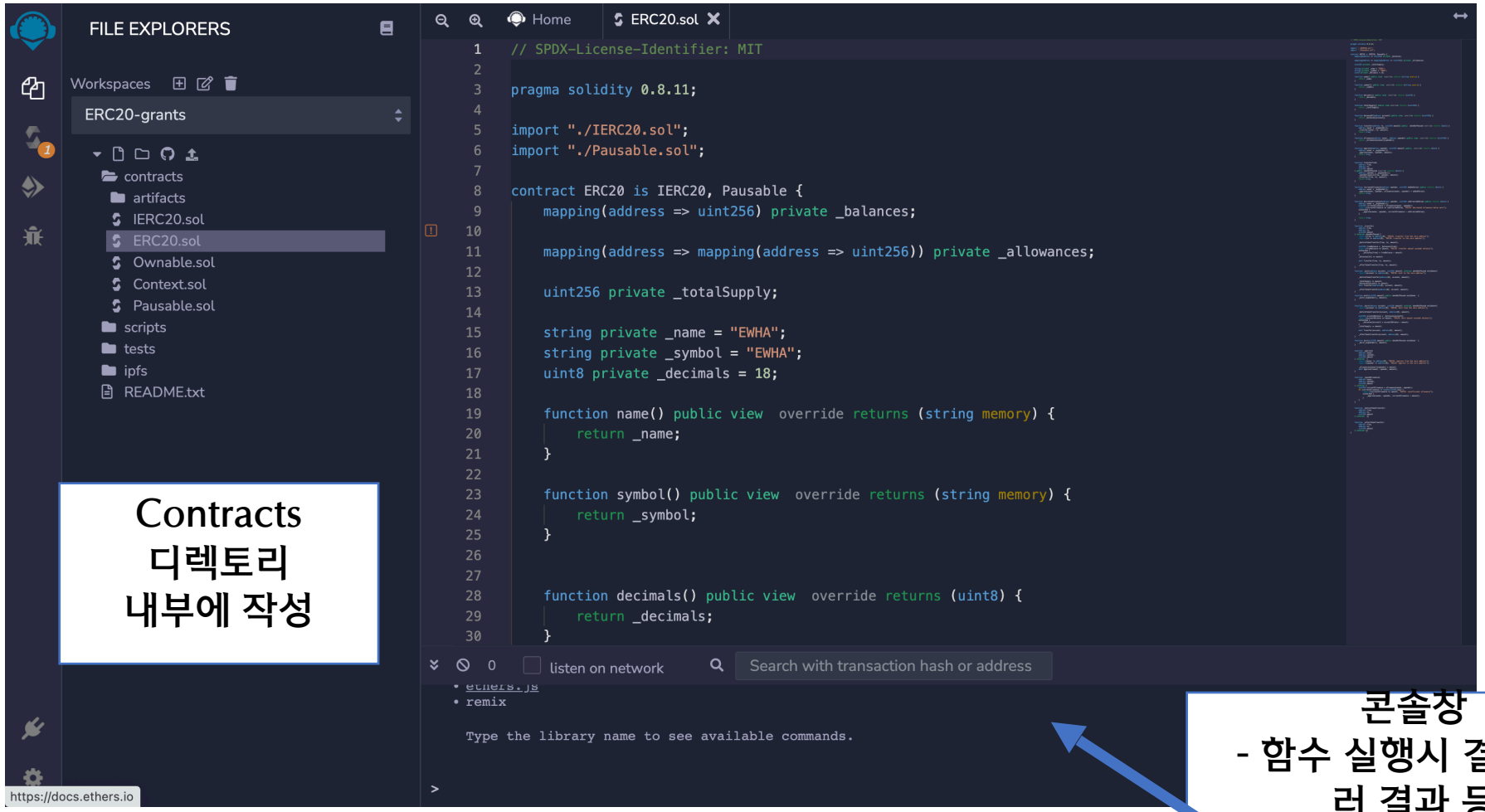
EWHA-CHAIN

Remix

Remix 

- Solidity 언어를 사용해 Ethereum Smart Contract를 컴파일 ,테스트, 디버깅, 배포하는 등의 통합 환경을 제공하는 툴
- 웹 브라우저를 통해 사용 가능
- http://remix.ethereum.org/

Remix로 컨트렉트 작성하기



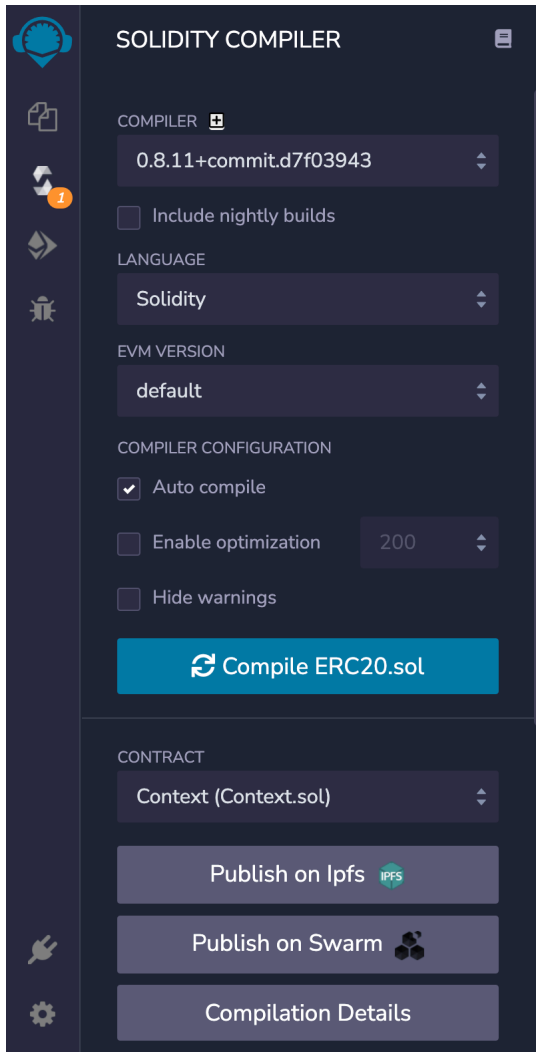
Contracts
디렉토리
내부에 작성

콘솔창
- 함수 실행시 결과, 에러 결과 등
다 출력됨

EWHA-CHAIN

Remix로 컨트렉트 컴파일

가장 에러를 잡기 편한 Remix!



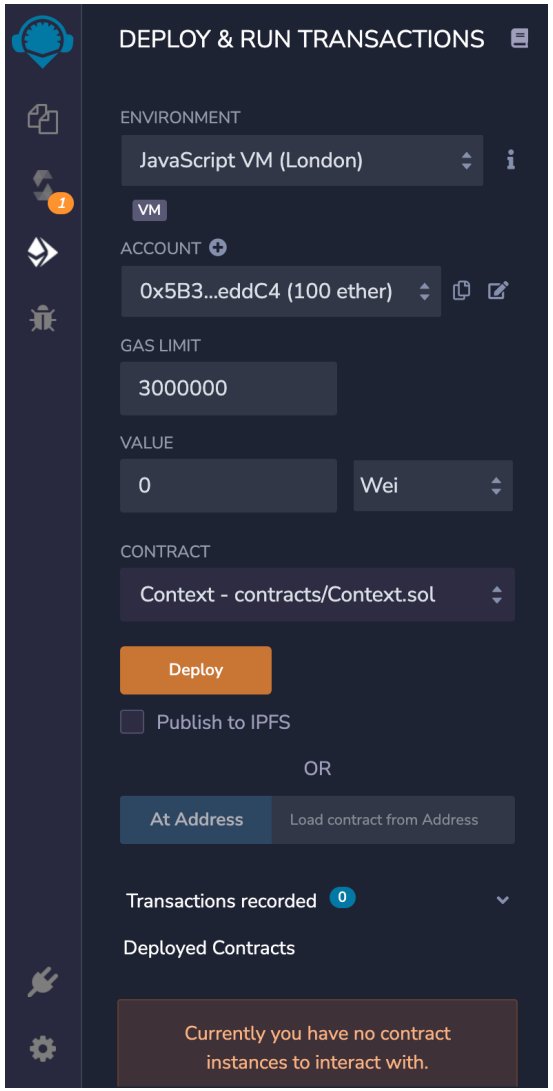
컴파일 버튼
(자동 컴파일 시 편리)

ABI 누르면 json 형식의 컨트렉트 정보 복사됨
(프론트엔드 연결 시 필요)

에러, 경고 등 뜬다

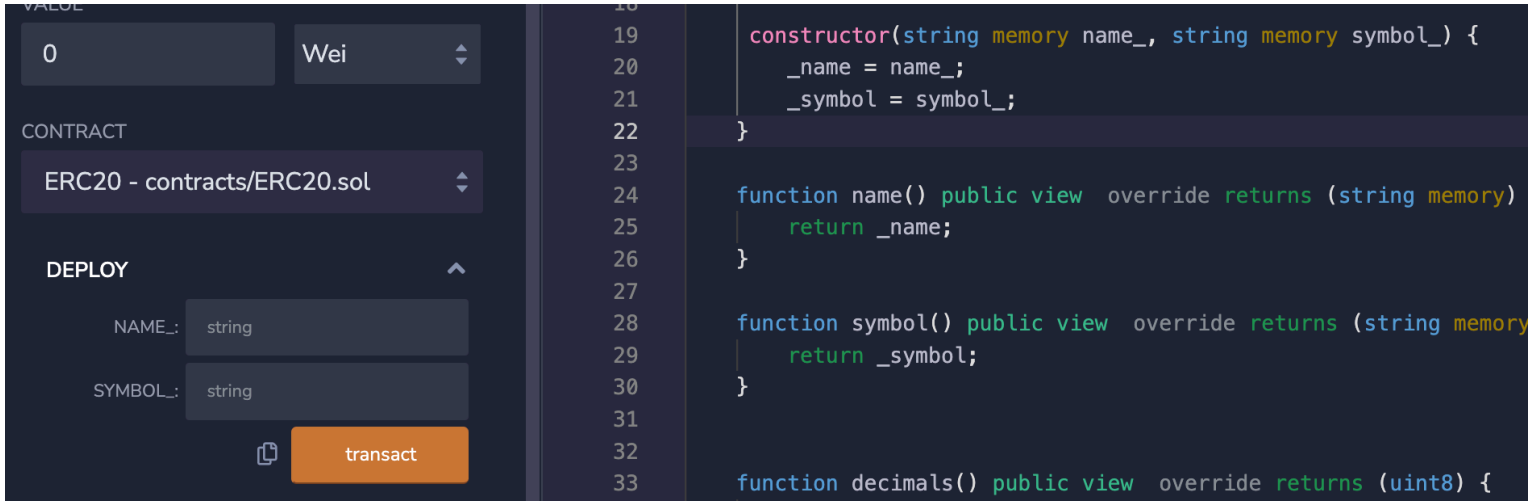
EWHA-CHAIN

Remix로 컨트렉트 로컬 배포하기



Deploy 버튼 누르기

** Contstructor가 있는 컨트렉트의 경우, Constructor에 필요한 인자가 들어가야 Deploy 가능



EWHA-CHAIN

추가 Solidity 학습

- CryptoZombie: https://cryptozombies.io
  -> 게임 요소가 있는 Solidity 학습 튜토리얼 웹
  (문법 학습을 위해 Solidity Path 섹션 반복학습 추천)
- Solidity Docs: https://docs.soliditylang.org/en/v0.8.13/
  -> 버전 업데이트가 잦아 Docs가 제일 정확
- 버전에 따라 문법이나 라이브러리가 판이하게 다른 경우들이 있음
  -> 책이나 강의의 경우 기본 코드 구조만 파악하고 문법 자체는 Docs를 보며 파악 권장
  -> 예시가 주제별로 잘 나와있음 : https://www.tutorialspoint.com/solidity/index.htm

EWHA-CHAIN