

MINT YOUR OWN TOKEN

SOPT WEB3 STUDY WEEK 2



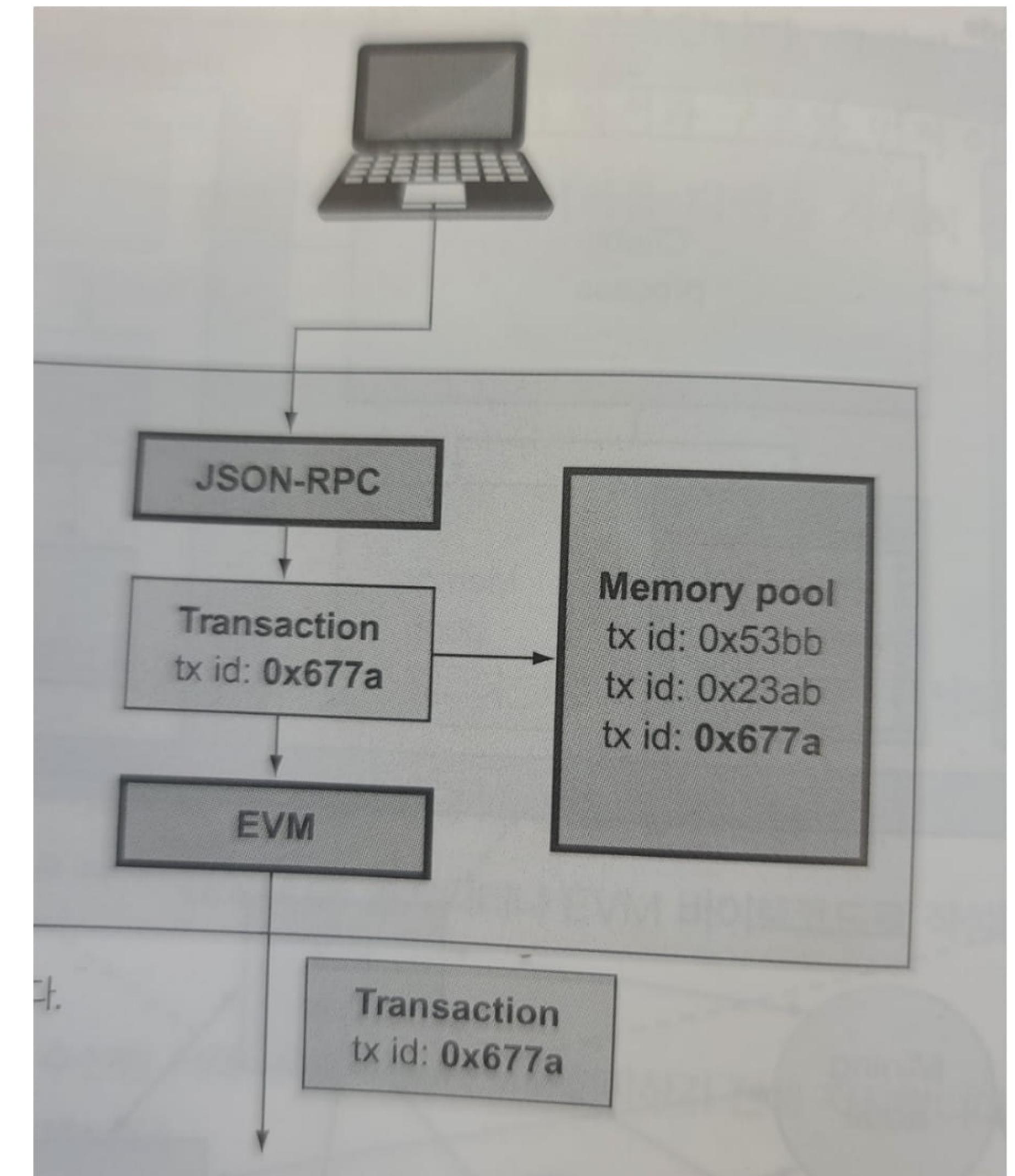
Dapp (탈중앙화 앱)

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

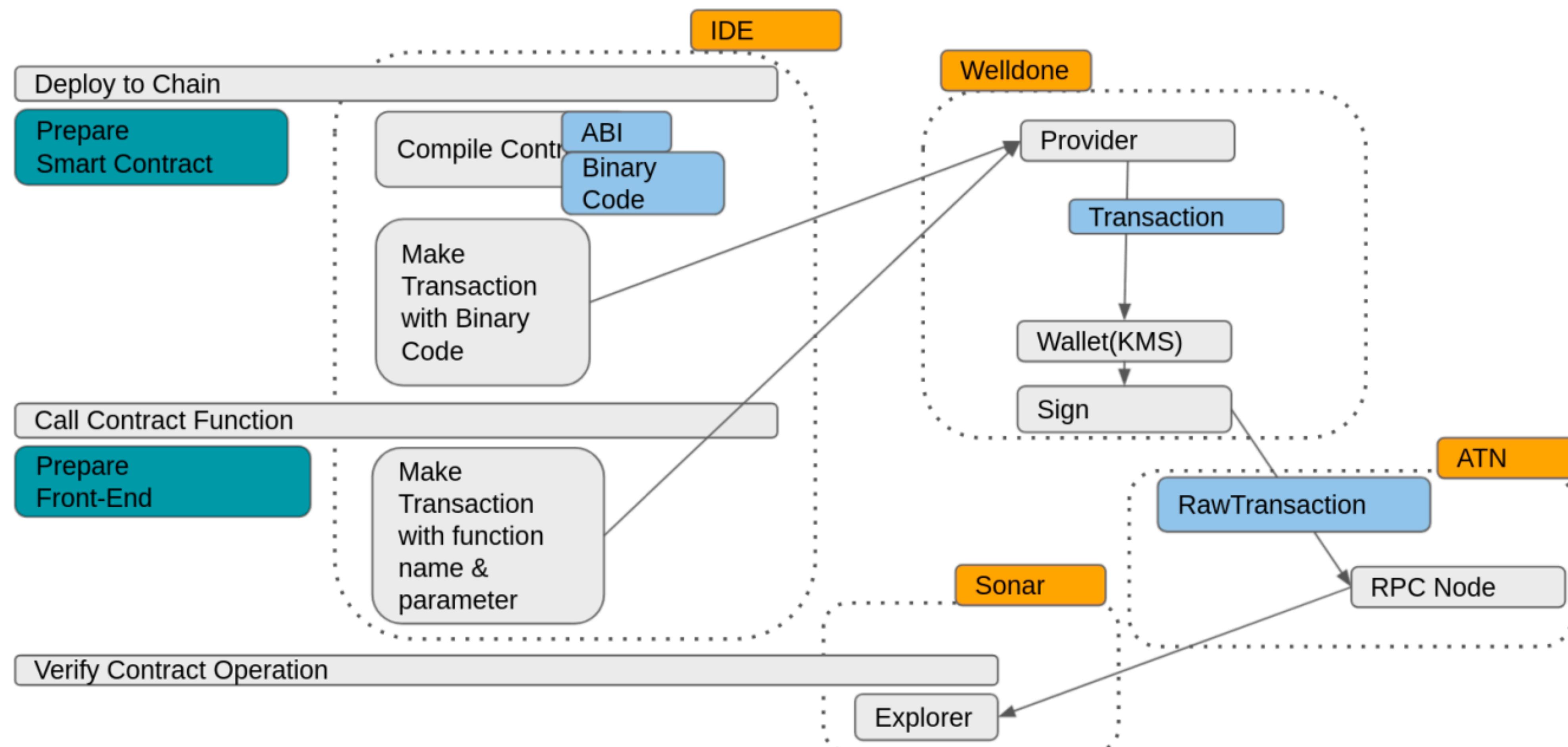
- 1) 사용자 이벤트는 웹 페이지의 js에서 처리합니다.
- 2) 이벤트 발생 시 web3.js 라이브러리 함수를 사용하여 스마트 컨트랙트와 연결된 이더리움 노드와 연결하고 이벤트에 맞는 함수 A를 호출합니다.
- 3) 연결된 로컬 이더리움 노드 (ex. 개발자 컴퓨터)는 메시지를 처리하고 검증한 후 피어 노드로 전달합니다.
- 4) 거래내역이 채굴 노드를 만날 때까지 전파합니다.
- 5) 각 노드는 새 block을 받으면 해당 block의 개별 거래내역이 정상적인지 + 전체 block이 유효한지 검증합니다. 이후 block에 존재하는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

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



How to building web3 application



Smart Contract (스마트 컨트렉트)

- 배포 과정
 - : Solidity로 작성 -> EVM 바이트 코드로 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)

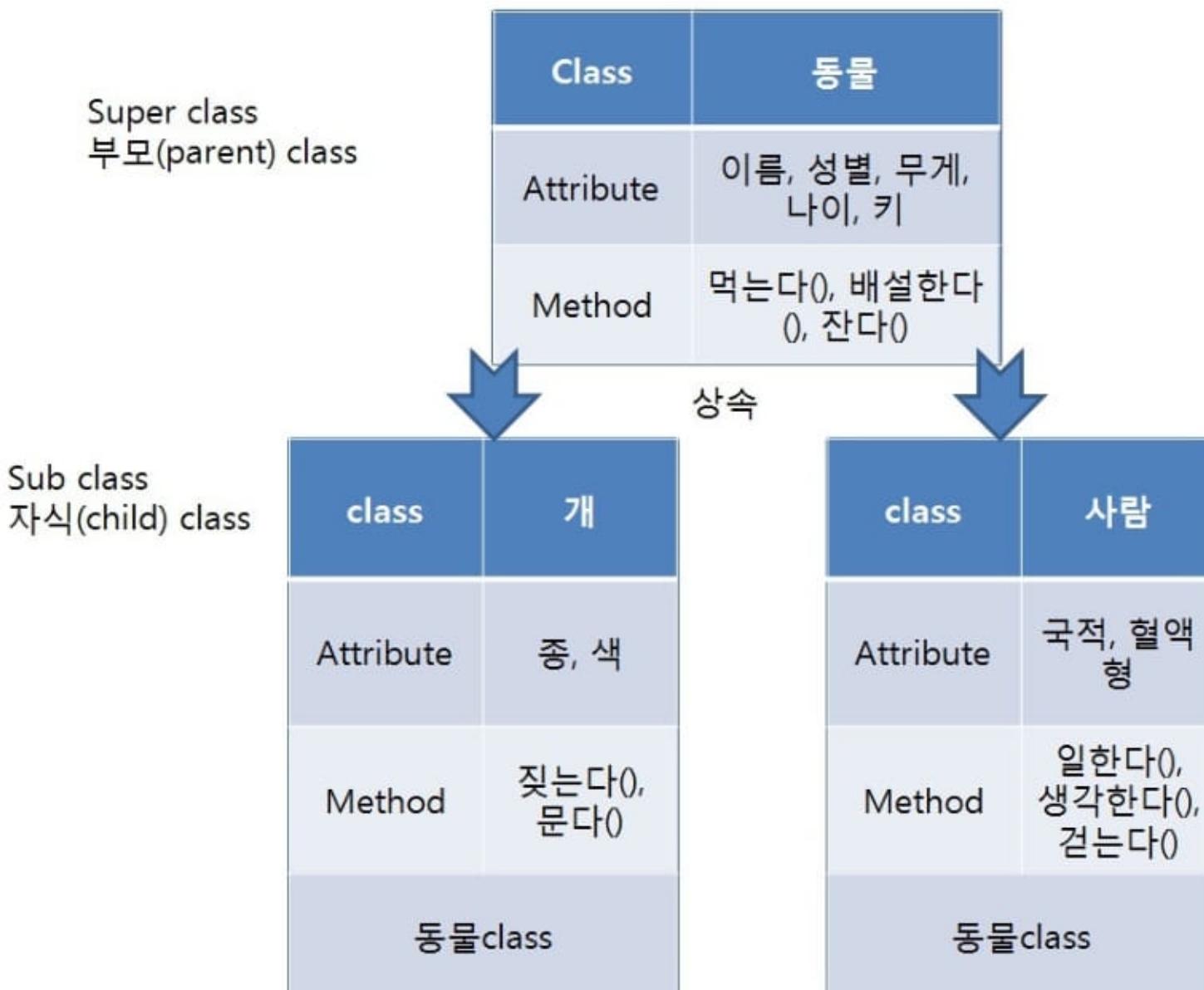
pragma

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.

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 {
```

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

```
// 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";
    }
}
```

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

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;

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)

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)

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;  
}
```

Function (1)

- 함수 선언
: function 함수명 (인자들 선언) {내용}
- 함수 호출
: 함수명 (인자들에 해당되는 값들)
- 함수 인자명은 위와 같이 언더스코어(_)로 시작해서, 함수 외부의 전역 변수와 구별하는 것이 관례
- Return 할 시, return 값을 정확히 명시

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

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

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

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;  
    }  
}
```

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 할수는 다른 컨트렉트에서 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 할수가 internal로 선언되었기 때문에 자식 컨트렉트에서 호출이 가능합니다.  
        eat();  
    }  
}
```

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");

modifier (제어자)

- 다른 함수들에 대한 제어를 위한 사용 함수
- 함수 실행 전의 요구사항 충족 여부를 확인하기 위한 목적
- Openzeppelin의 onlyOwner : 컨트렉트의 소유자 (컨트렉트를 배포한 사람이 기본)

```
// 사용자의 나이를 저장하기 위한 맵핑
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
```

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

비교, 논리 연산자

• 비교 연산자

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

• 논리 연산자

<code>&&</code>	두 조건이 모두 참이면 true를 반환하고 하나 또는 둘 다 거짓이면 false를 반환
<code> </code>	하나 또는 두 조건 모두 참이면 true를 반환하고 둘 다 거짓이면 false를 반환
<code>!</code>	true를 false로 false를 true로 반환

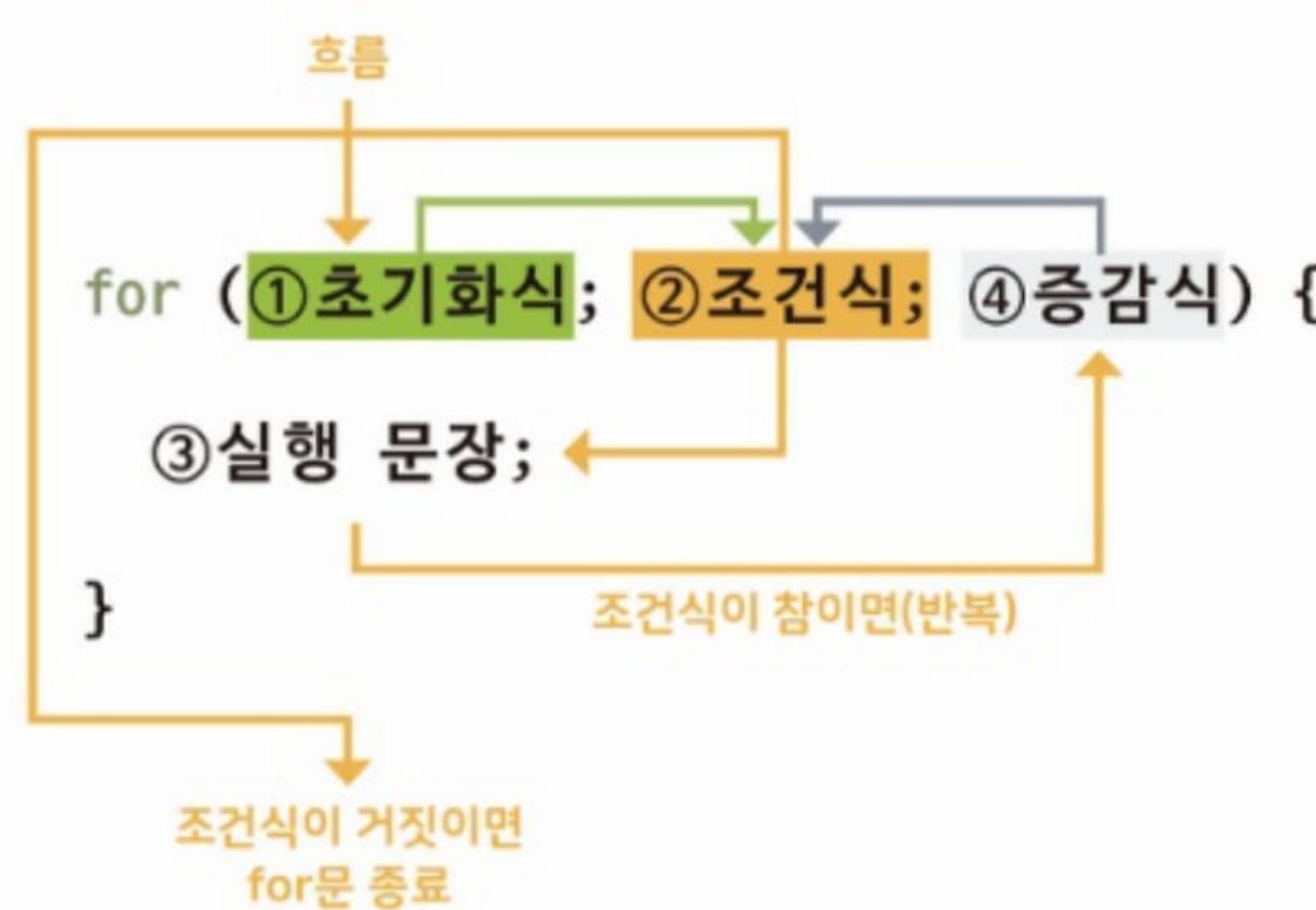
조건문 - If, else if, else

- 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);
}
```

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] );
        }
    }
}
```

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;
```

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)

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);

}
```

이벤트

- 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

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.8.0 <0.9.0;

contract lec13 {

    event info(string name, uint256 money);

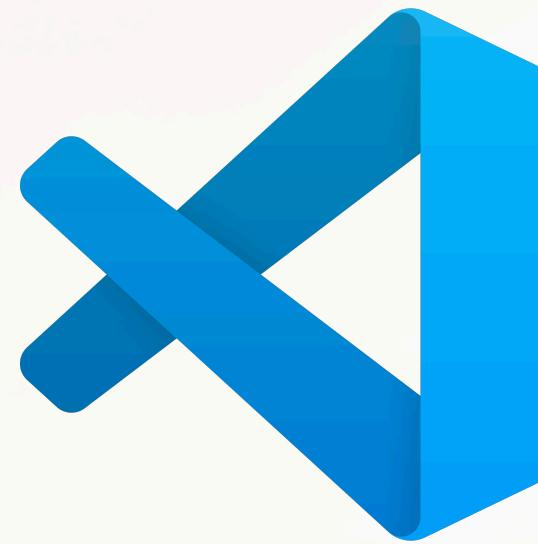
    function sendMoney() public {
        emit info("KimDaeJin", 1000);
    }
}
```

기타 솔리디티 개념들

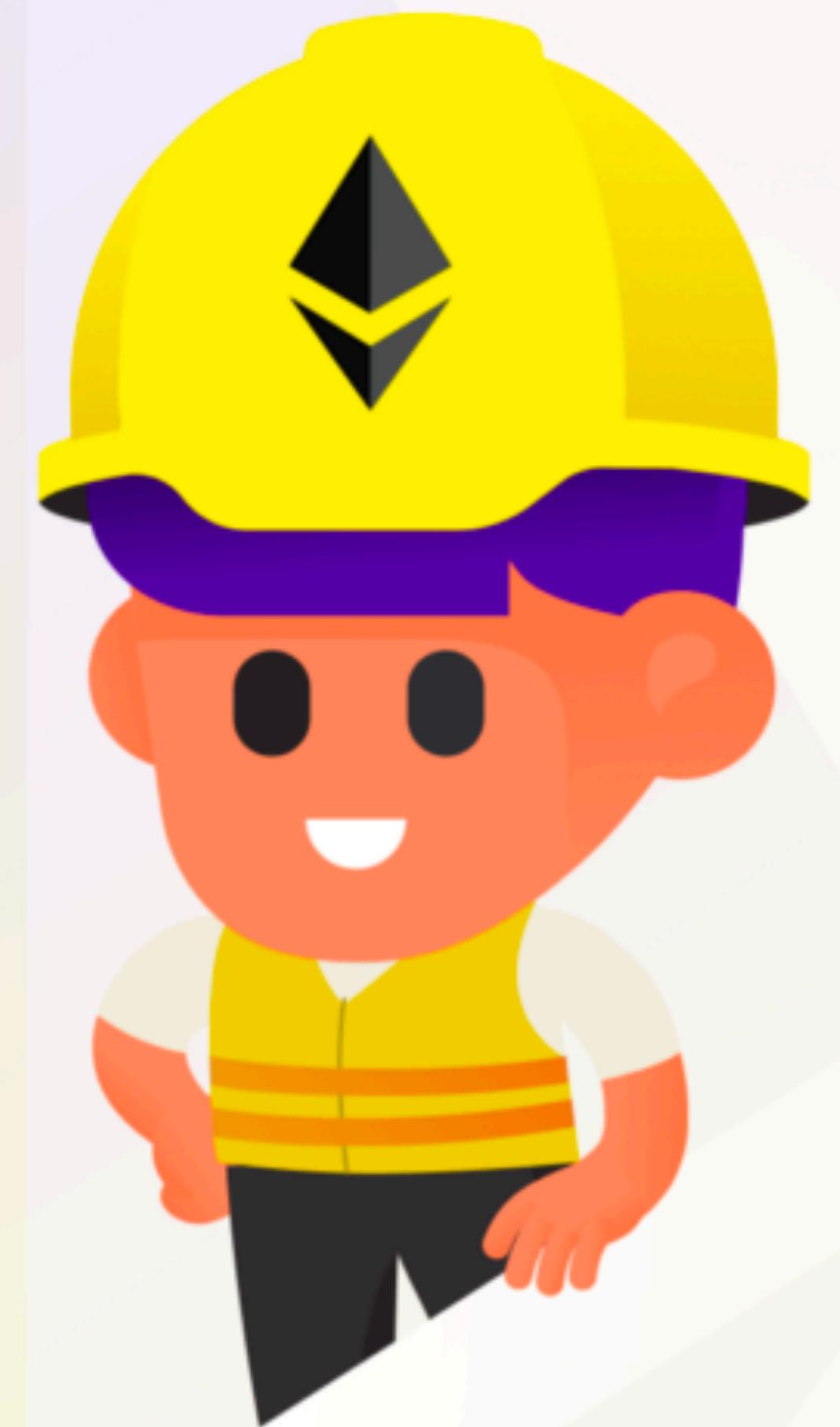
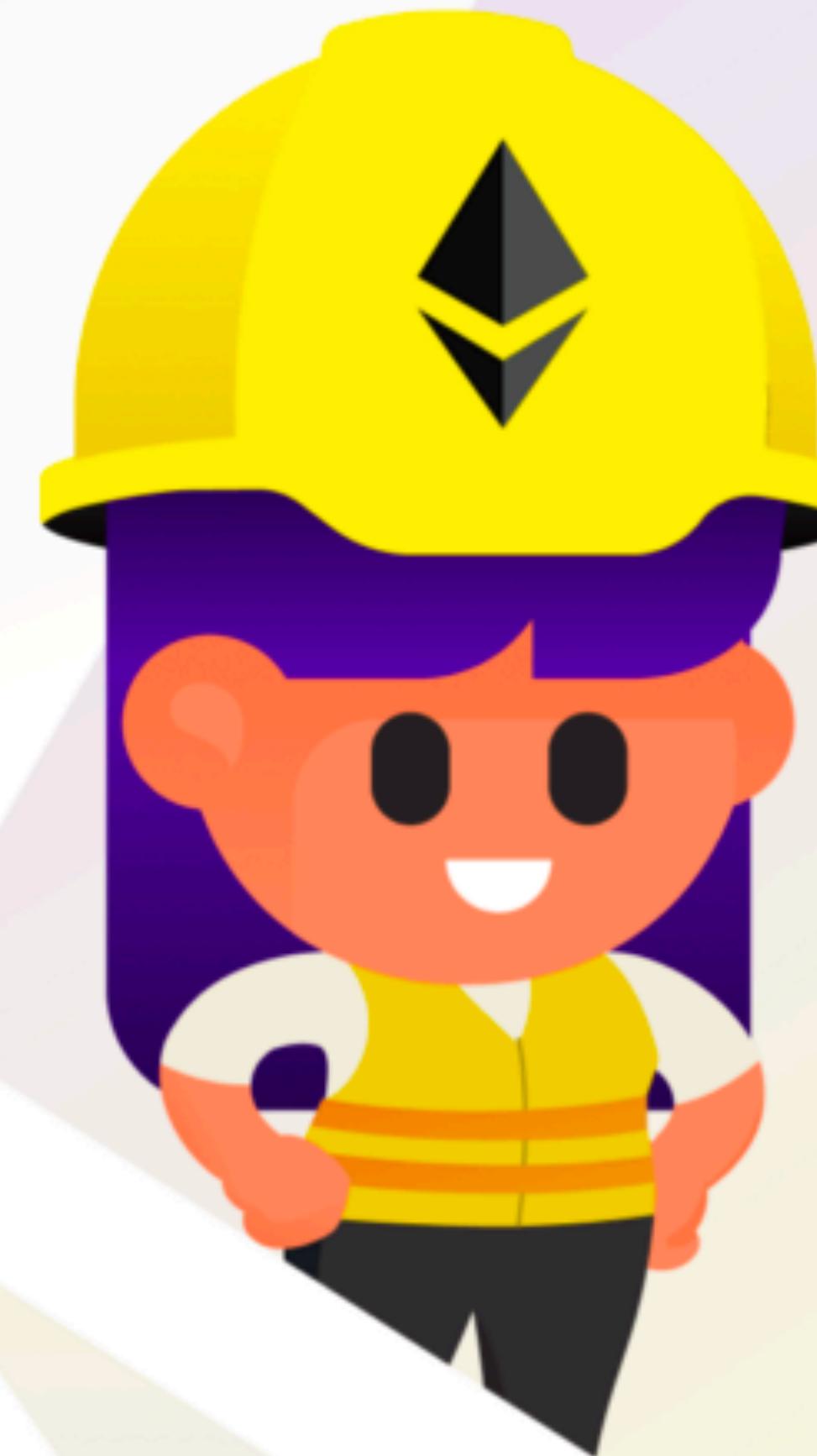
- payable 키워드: 함수에서 Ether을 지불 가능하도록 명시
ex) function getStockPRice(string_stockTicker) payable returns~
- keccak : 난수 생성 함수

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

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



Hardhat



```
npm init
```

```
npm install --save-dev hardhat
```

```
npx hardhat
```

```
npm install --save-dev @nomiclabs/hardhat-waffle ethereum-waffle chai @nomiclabs/hardhat-ethers ethers #
```

```
$ npx hardhat
888 888 888 888 888
888 888 888 888 888
888 888 888 888 888
8888888888 8888b. 888d888 .d88888 88888b. 8888b. 888888
888 888 "88b 888P" d88" 888 888 "88b "88b 888
888 888 .d888888 888 888 888 888 .d888888 888
888 888 888 888 888 Y88b 888 888 888 888 888 Y88b.
888 888 "Y888888 888 "Y88888 888 888 "Y888888 "Y888
```

```
Welcome to Hardhat v2.0.8
```

```
? What do you want to do? ...
> Create a sample project
    Create an advanced sample project
    Create an advanced sample project that uses TypeScript
    Create an empty hardhat.config.js
    Quit
```

```
require("@nomiclabs/hardhat-waffle");

// This is a sample Hardhat task. To learn how to create your own go to
// https://hardhat.org/guides/create-task.html
task("accounts", "Prints the list of accounts", async (taskArgs, hre) => {
    const accounts = await hre.ethers.getSigners();

    for (const account of accounts) {
        console.log(account.address);
    }
});

// You need to export an object to set up your config
// Go to https://hardhat.org/config/ to learn more

/**
 * @type import('hardhat/config').HardhatUserConfig
 */
module.exports = {
    solidity: "0.8.4",
};
```

hardhat.config.js

```
$ npx hardhat accounts  
0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266  
0x70997970C51812dc3A010C7d01b50e0d17dc79C8  
0x3C44CdDdB6a900fa2b585dd299e03d12FA4293BC  
0x90F79bf6EB2c4f870365E785982E1f101E93b906  
0x15d34AAf54267DB7D7c367839AAf71A00a2C6A65  
0x9965507D1a55bcc2695C58ba16FB37d819B0A4dc  
0x976EA74026E726554dB657fA54763abd0C3a0aa9  
0x14dC79964da2C08b23698B3D3cc7Ca32193d9955  
0x23618e81E3f5cdF7f54C3d65f7FBc0aBf5B21E8f  
0xa0Ee7A142d267C1f36714E4a8F75612F20a79720  
0xBcd4042DE499D14e55001CcbB24a551F3b954096  
0x71bE63f3384f5fb98995898A86B02Fb2426c5788  
0xFABB0ac9d68B0B445fB7357272Ff202C5651694a  
0x1CBd3b2770909D4e10f157cABC84C7264073C9Ec  
0xdF3e18d64BC6A983f673Ab319CCaE4f1a57C7097  
0xcd3B766CCDd6AE721141F452C550Ca635964ce71  
0x2546BcD3c84621e976D8185a91A922aE77ECEc30  
0xbDA5747bFD65F08deb54cb465eB87D40e51B197E  
0xD2FD4581271e230360230F9337D5c0430Bf44C0  
0x8626f6940E2eb28930eFb4CeF49B2d1F2C9C1199
```

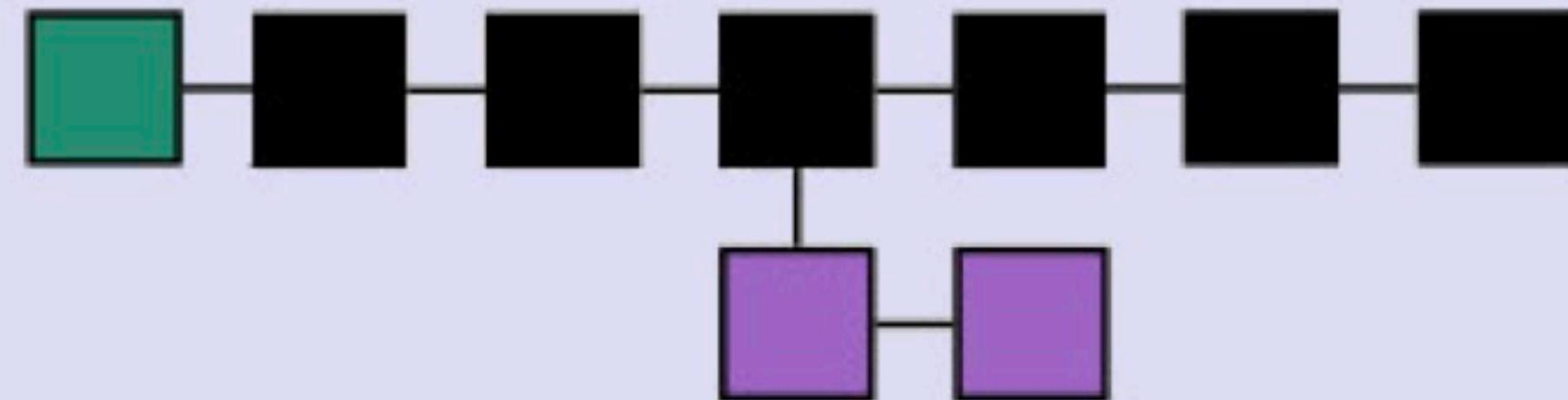




```
module.exports = {
  solidity: {
    compilers: [
      {
        version: "0.8.10"
      },
      {
        version: "0.7.6",
        settings: {}
      }
    ]
  },
  networks: {
    // mumbai: {
    //   url: "https://polygon-mumbai.g.alchemy.com/v2/" + process.env.ALCHEMY_MUMBAI_KEY,
    //   accounts: [process.env.METAMASK_PRIVATE_KEY],
    //   gas: 9100000,
    //   gasPrice: 8000000000
    // },
    // ropsten: {
    //   url: "https://ethereum-ropsten-rpc.allthatnode.com/" + process.env.DSRV_ROPSTEN_KEY,
    //   accounts: [process.env.METAMASK_PRIVATE_KEY],
    //   gas: 9100000,
    //   gasPrice: 8000000000
    // },
    // rinkeby: {
    //   url: "https://ethereum-rinkeby-rpc.allthatnode.com/" + process.env.DSRV_ROPSTEN_KEY,
    //   accounts: [process.env.METAMASK_PRIVATE_KEY],
    //   gas: 9100000,
    //   gasPrice: 8000000000
    // },
    polygon: {
      url: "https://polygon-mainnet.g.alchemy.com/v2/" + process.env.ALCHEMY_API_KEY,
      accounts: [process.env.METAMASK_PRIVATE_KEY]
    }
  },
  etherscan: {
    apiKey: process.env.ETHERSCAN_KEY
  }
};
```

BLOCKCHAIN TUTORIAL 25

Testnet & Faucets





openZeppelin

```
$ npm install @openzeppelin/contracts
```

Counters

#

Provides counters that can only be incremented or decremented by one. This can be used e.g. to track the number of elements in a mapping, issuing ERC721 ids, or counting request ids.

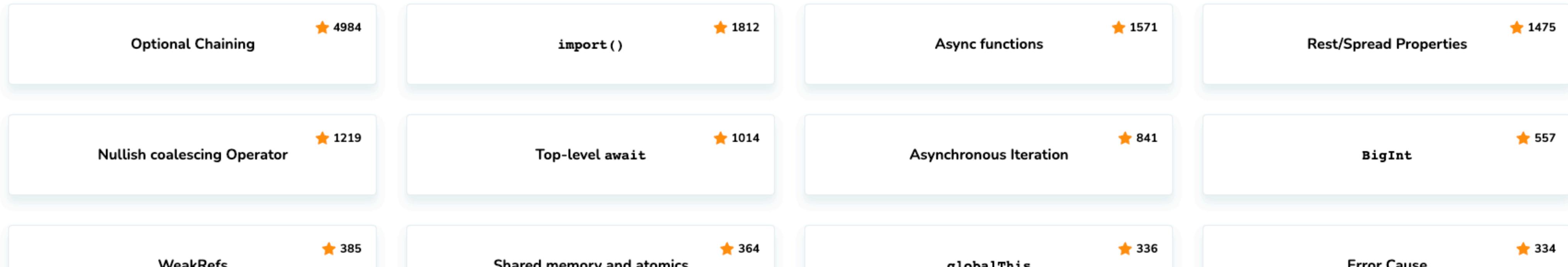
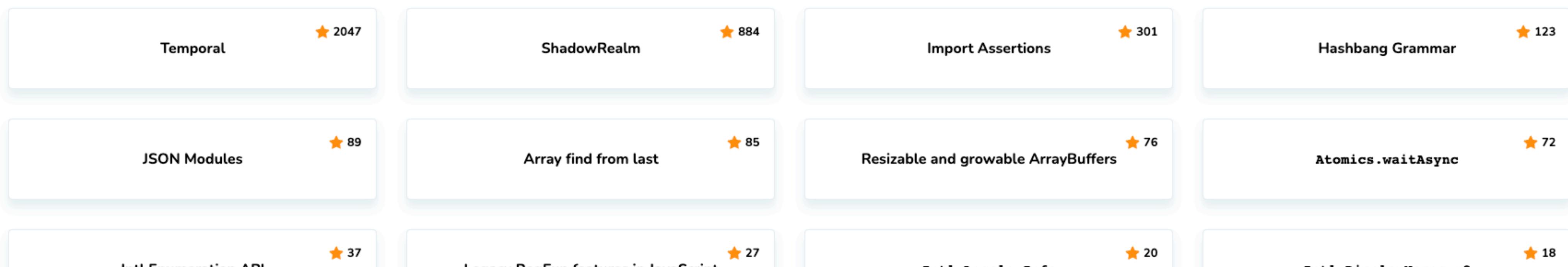
Include with `using Counters for Counters.Counter;` Since it is not possible to overflow a 256 bit integer with increments of one, `increment` can skip the [SafeMath](#) overflow check, thereby saving gas. This does assume however correct usage, in that the underlying `_value` is never directly accessed.

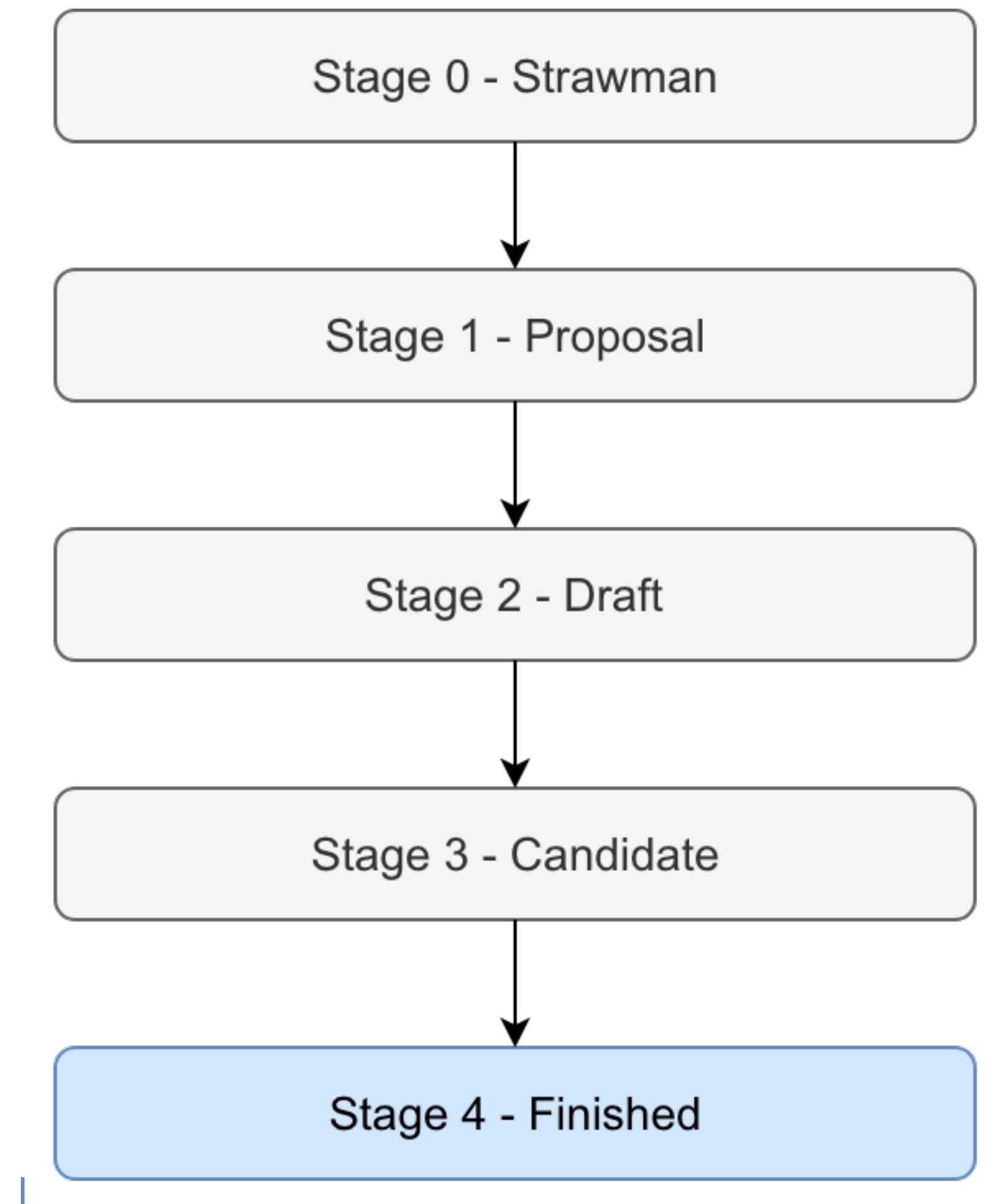
FUNCTIONS

`current(counter)`

`increment(counter)`

`decrement(counter)`

Search for proposals...Stage 4 (Finished)Stage 3 (Candidate)<https://www.proposals.es/>



Ecma 인터내셔널의 여러 기술 위원회(Technial Committee, 이하 TC) 중 ECMA-262 명세의 관리는 TC39 위원회가 맡고 있다.

TC39의 구성원 목록은 Mozilla, Google, Apple, Microsoft 등의 메이저 브라우저 벤더를 비롯해 Facebook, Twitter 등의 다양한 단체로 이루어져 있다. 대부분의 구성원이 표준을 올바르게 구현해야 할 책임을 갖는, 언어 표준의 변화에 직접적으로 영향을 받는 단체다.

엄밀히는 기업/단체가 구성원이지만, 'TC39 구성원'이라는 용어가 해당 기업/단체가 보낸 대리인을 지칭하는 경우도 많다.

<https://github.com/tc39/tc39-notes>

<https://ahnheejong.name/articles/ecmascript-tc39/>

Ethereum Improvement Proposals

All Core Networking Interface ERC Meta Informational

EIPs [gitter](#) [join chat](#) [rss](#) [Last Calls](#)

Ethereum Improvement Proposals (EIPs) describe standards for the Ethereum platform, including core protocol specifications, client APIs, and contract standards. Network upgrades are discussed separately in the [Ethereum Project Management](#) repository.

Contributing

First review [EIP-1](#). Then clone the repository and add your EIP to it. There is a [template EIP here](#). Then submit a Pull Request to Ethereum's EIPs repository.

EIP status terms

- **Idea** - An idea that is pre-draft. This is not tracked within the EIP Repository.
- **Draft** - The first formally tracked stage of an EIP in development. An EIP is merged by an EIP Editor into the EIP repository when properly formatted.
- **Review** - An EIP Author marks an EIP as ready for and requesting Peer Review.
- **Last Call** - This is the final review window for an EIP before moving to FINAL. An EIP editor will assign Last Call status and set a review end date ('last-call-deadline'), typically 14 days later. If this period results in necessary normative changes it will revert to Draft.
- **Final** - This EIP represents the final standard. A Final EIP exists in a state of finality and should only be updated to correct errata and add non-normative clarifications.
- **Stagnant** - Any EIP in Draft or Review if inactive for a period of 6 months or greater is moved to Stagnant. An EIP may be resurrected from this state by Authors or EIP Editors through moving it back to Draft.
- **Withdrawn** - The EIP Author(s) have withdrawn the proposed EIP. This state has finality and can no longer be resurrected using this EIP number. If the idea is pursued at later date it is considered a new proposal.
- **Living** - A special status for EIPs that are designed to be continually updated and not reach a state of finality. This includes most notably EIP-1.

EIP Types

EIPs are separated into a number of types, and each has its own list of EIPs.

Standard Track (407)

Describes any change that affects most or all Ethereum implementations, such as a change to the network protocol, a change in block or transaction validity rules, proposed application standards/conventions, or any change or addition that affects the interoperability of Ethereum. Furthermore Standard EIPs can be broken down into the following categories.

Core (179)

Improvements requiring a consensus fork (e.g. [EIP-5](#), [EIP-101](#)), as well as changes that are not necessarily consensus critical but may be relevant to "core dev" discussions (for example, the miner/node strategy changes 2, 3, and 4 of [EIP-86](#)).

Networking (12)

Includes improvements around devp2p ([EIP-8](#)) and Light Ethereum Subprotocol, as well as proposed improvements to network protocol specifications of whisper and swarm.

Interface (41)

Regularly, a committee of editors, including members of the Ethereum community and Ethereum Foundation, review new EIPs. While the editors used to include the founder of Ethereum, Vitalik Buterin, the team has evolved with the cryptocurrency.

The team consists of Micah Zoltu, Nick Savers, Nick Johnson, Hudson James, Matt Garnett, Casey Detrio, Greg Colvin, and Alex Beregszaszi. An EIP then goes through a number of statuses before upgrade implementation. This may include, as part of the process, further drafts that require examination before approval and implementation.

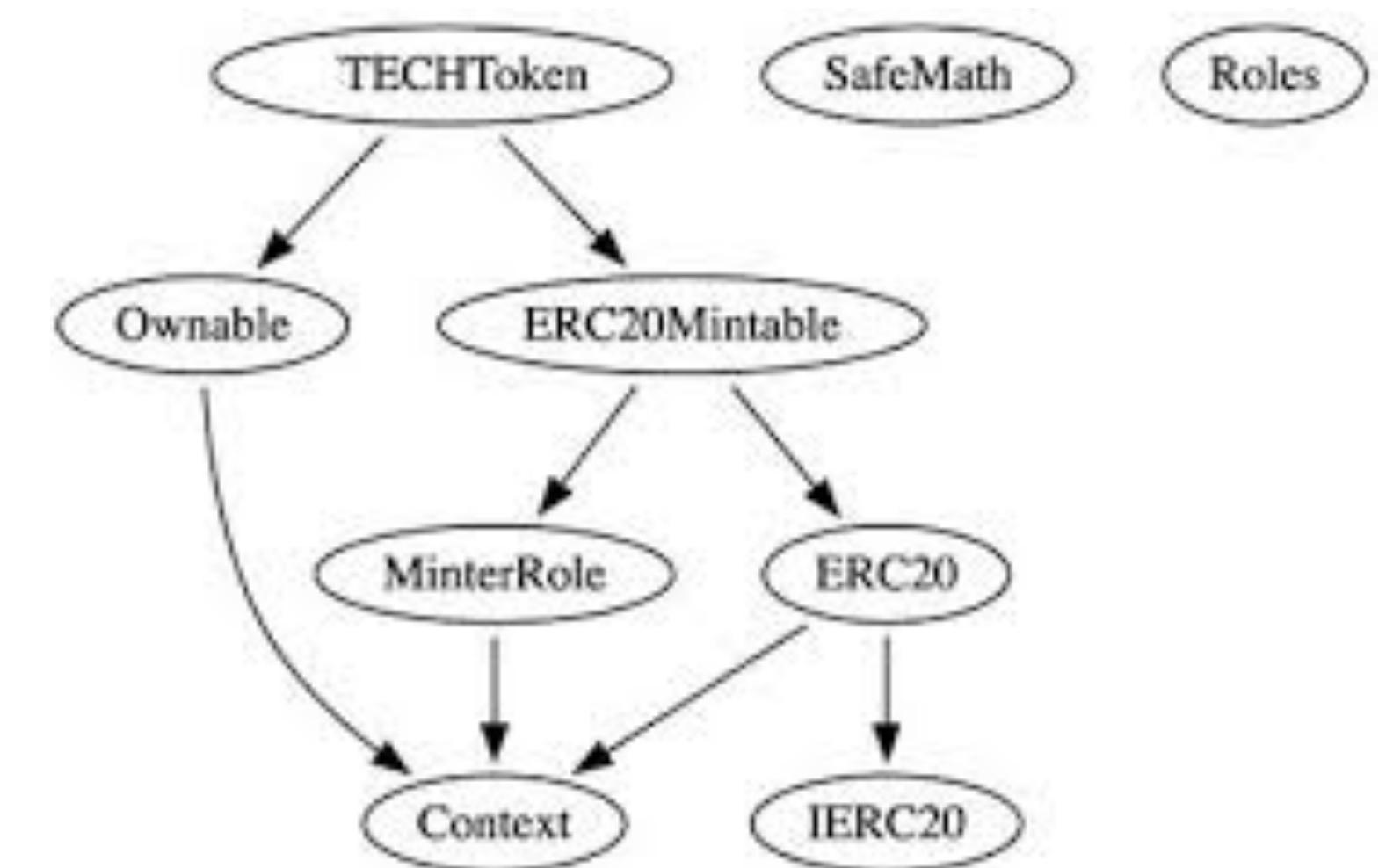
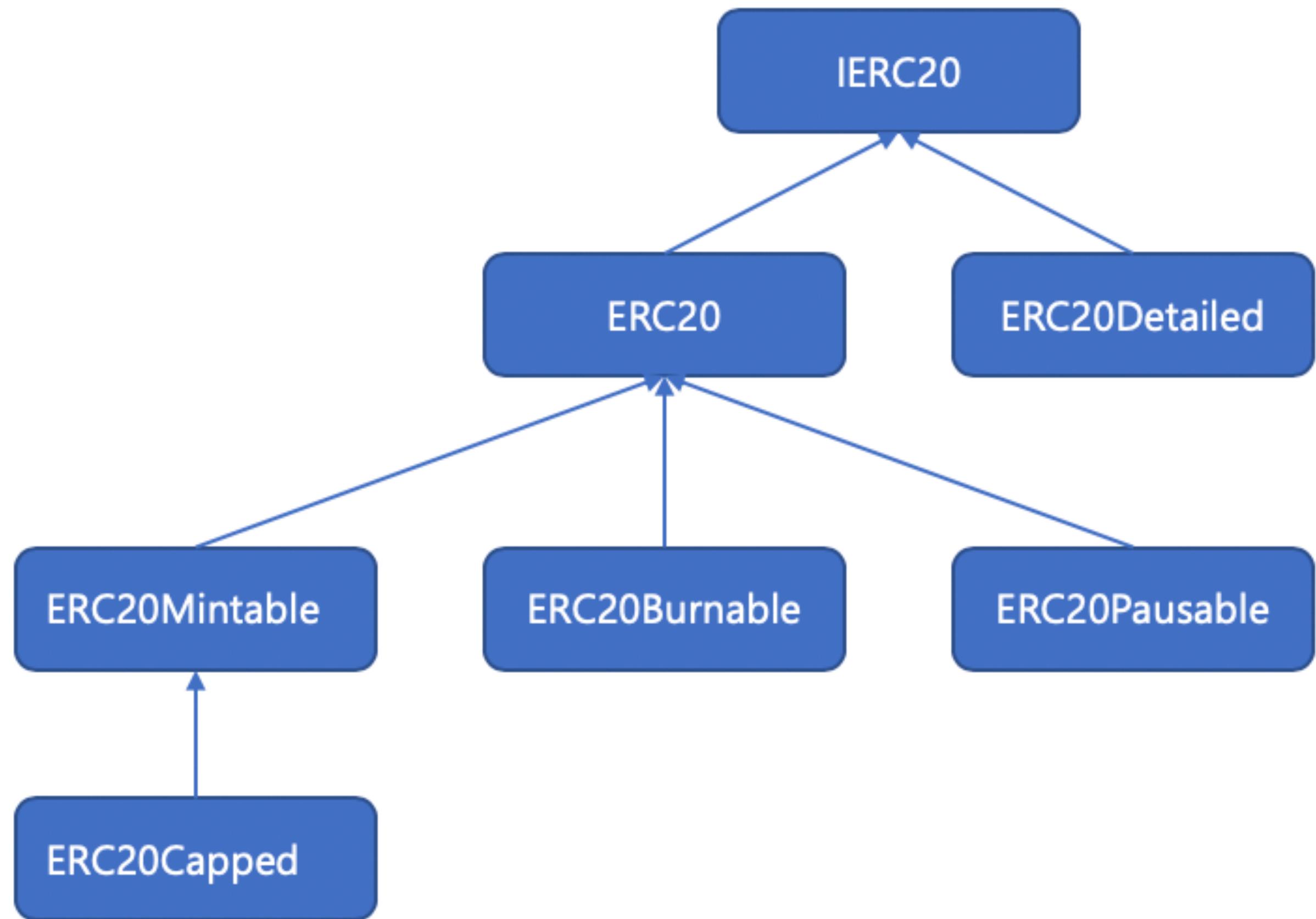
<https://github.com/ethereum/EIPs>

<https://eips.ethereum.org/>

표 3. ERC20과 ERC721 코드레벨단 차이

기능	ERC	입력 값	진행
Mint (발행)	ERC20	주소, 발행량	주소로 발행량 전송
	ERC721	받는 주소, 토큰ID	토큰ID의 소유자를 수신 주소로 변경 & 전송
Transfer (전송)	ERC20	보내는 주소, 받는 주소, 보내는 양	보내는 주소에서 송신 주소로 양만큼 토큰 전송
	ERC721	보내는 주소, 받는 주소, 토큰ID	1. 권한 처리 2. 토큰ID의 소유자를 받는 주소로 변경 3. 토큰 전송
Burn (소각)	ERC20	주소, 소각할 양	주소에서 양만큼 소각
	ERC721	토큰ID	1. 권한 처리 2. 토큰ID의 소유자 정보 삭제 3. 토큰ID 소각

자료: SPRi, 미래에셋증권 리서치센터



```
ERC20.sol

/** @dev Creates `amount` tokens and assigns them to `account`, increasing
 * the total supply.
 *
 * Emits a {Transfer} event with `from` set to the zero address.
 *
 * Requirements:
 *
 * - `account` cannot be the zero address.
 */
function _mint(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: mint to the zero address");

    _beforeTokenTransfer(address(0), account, amount);

    _totalSupply += amount;
    _balances[account] += amount;
    emit Transfer(address(0), account, amount);

    _afterTokenTransfer(address(0), account, amount);
}
```



The image shows a screenshot of a code editor with a dark theme, displaying a Solidity smart contract named `ERC20.sol`. The code is annotated with detailed comments explaining the function's purpose, requirements, and internal logic.

```
ERC20.sol

/*
 * @dev Moves `amount` of tokens from `sender` to `recipient`.
 *
 * This internal function is equivalent to {transfer}, and can be used to
 * e.g. implement automatic token fees, slashing mechanisms, etc.
 *
 * Emits a {Transfer} event.
 *
 * Requirements:
 *
 * - `sender` cannot be the zero address.
 * - `recipient` cannot be the zero address.
 * - `sender` must have a balance of at least `amount`.
 */
function _transfer(
    address sender,
    address recipient,
    uint256 amount
) internal virtual {
    require(sender != address(0), "ERC20: transfer from the zero address");
    require(recipient != address(0), "ERC20: transfer to the zero address");

    _beforeTokenTransfer(sender, recipient, amount);

    uint256 senderBalance = _balances[sender];
    require(senderBalance >= amount, "ERC20: transfer amount exceeds balance");
    unchecked {
        _balances[sender] = senderBalance - amount;
    }
    _balances[recipient] += amount;

    emit Transfer(sender, recipient, amount);

    _afterTokenTransfer(sender, recipient, amount);
}
```

Events

Events allow logging to the Ethereum blockchain. Some use cases for events are:

- Listening for events and updating user interface
- A cheap form of storage

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.10;

contract Event {
    // Event declaration
    // Up to 3 parameters can be indexed.
    // Indexed parameters helps you filter the logs by the indexed parameter
    event Log(address indexed sender, string message);
    event AnotherLog();

    function test() public {
        emit Log(msg.sender, "Hello World!");
        emit Log(msg.sender, "Hello EVM!");
        emit AnotherLog();
    }
}
```

Payable

Functions and addresses declared `payable` can receive `ether` into the contract.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.10;

contract Payable {
    // Payable address can receive Ether
    address payable public owner;

    // Payable constructor can receive Ether
    constructor() payable {
        owner = payable(msg.sender);
    }

    // Function to deposit Ether into this contract.
    // Call this function along with some Ether.
    // The balance of this contract will be automatically updated.
    function deposit() public payable {

        // Call this function along with some Ether.
        // The function will throw an error since this function is not payable.
        function notPayable() public {}

        // Function to withdraw all Ether from this contract.
        function withdraw() public {
            // get the amount of Ether stored in this contract
            uint amount = address(this).balance;

            // send all Ether to owner
            // Owner can receive Ether since the address of owner is payable
            (bool success, ) = owner.call{value: amount}("");
            require(success, "Failed to send Ether");
        }

        // Function to transfer Ether from this contract to address from input
        function transfer(address payable _to, uint _amount) public {
            // Note that "to" is declared as payable
            (bool success, ) = _to.call{value: _amount}("");
            require(success, "Failed to send Ether");
        }
    }
}
```

Ownable

🔗 #

```
import "@openzeppelin/contracts/access/Ownable.sol";
```

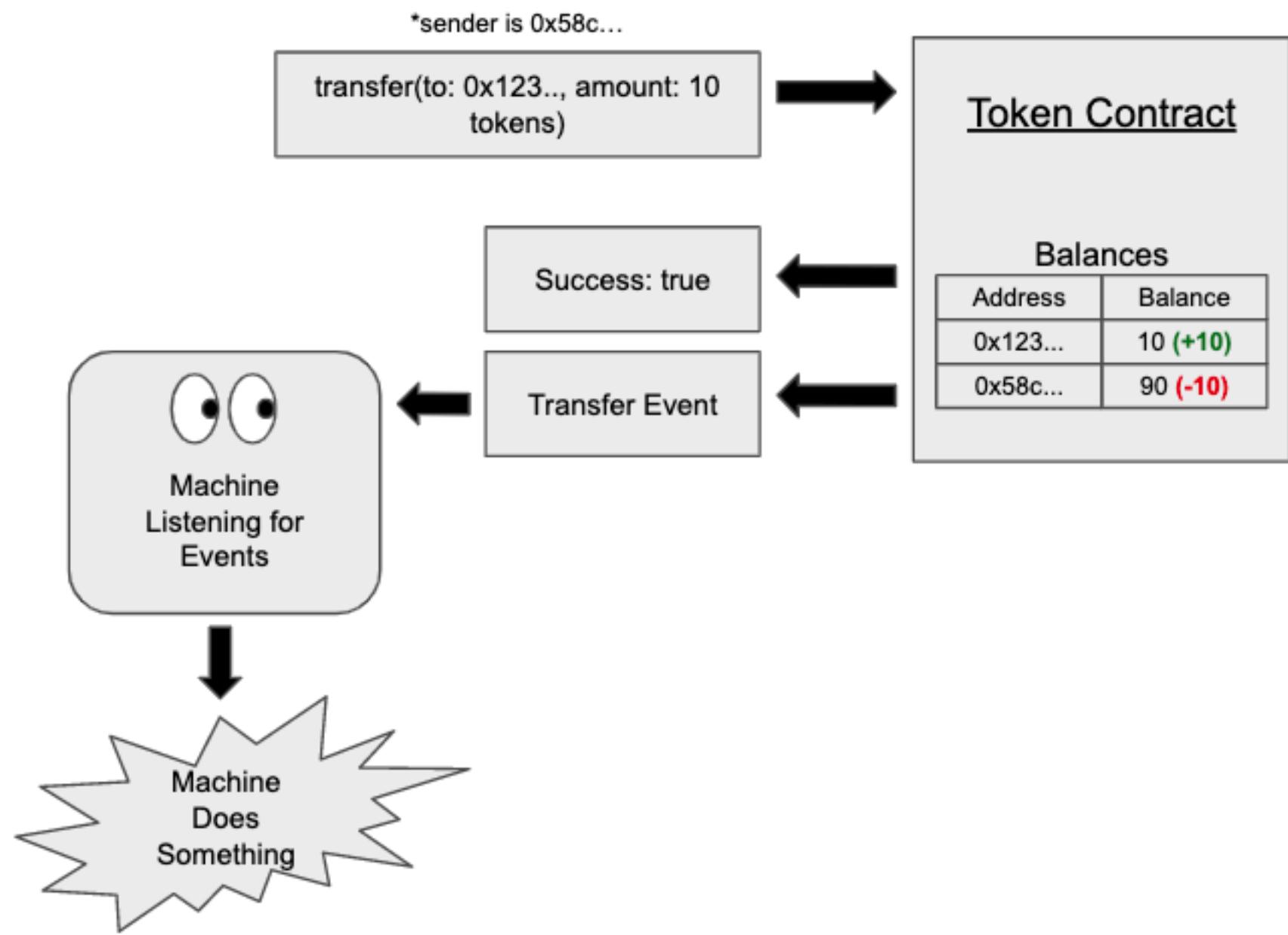
Contract module which provides a basic access control mechanism, where there is an account (an owner) that can be granted exclusive access to specific functions.

By default, the owner account will be the one that deploys the contract. This can later be changed with [transferOwnership](#).

This module is used through inheritance. It will make available the modifier `onlyOwner`, which can be applied to your functions to restrict their use to the owner.

MODIFIERS

`onlyOwner()`



`transfer(address to, uint256 amount) → bool`

Moves `amount` tokens from the caller's account to `to`.

Returns a boolean value indicating whether the operation succeeded.

Emits a [Transfer](#) event.

Sending Ether (transfer, send, call)

How to send Ether?

You can send Ether to other contracts by

- `transfer` (2300 gas, throws error)
- `send` (2300 gas, returns bool)
- `call` (forward all gas or set gas, returns bool)

How to receive Ether?

A contract receiving Ether must have at least one of the functions below

- `receive() external payable`
- `fallback() external payable`

`receive()` is called if `msg.data` is empty, otherwise `fallback()` is called.

Which method should you use?

`call` in combination with re-entrancy guard is the recommended method to use after December 2019.

Guard against re-entrancy by

- making all state changes before calling other contracts
- using re-entrancy guard modifier

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.10;

contract ReceiveEther {
    /*
    Which function is called, fallback() or receive()?
        send Ether
        |
        msg.data is empty?
        / \
        yes  no
        / \
receive() exists?  fallback()
        / \
        yes  no
        / \
receive()  fallback()
*/
// Function to receive Ether. msg.data must be empty
receive() external payable {}

// Fallback function is called when msg.data is not empty
fallback() external payable {}

function getBalance() public view returns (uint) {
    return address(this).balance;
}
}
```

```
contract SendEther {
    function sendViaTransfer(address payable _to) public payable {
        // This function is no longer recommended for sending Ether.
        _to.transfer(msg.value);
    }

    function sendViaSend(address payable _to) public payable {
        // Send returns a boolean value indicating success or failure.
        // This function is not recommended for sending Ether.
        bool sent = _to.send(msg.value);
        require(sent, "Failed to send Ether");
    }

    function sendViaCall(address payable _to) public payable {
        // Call returns a boolean value indicating success or failure.
        // This is the current recommended method to use.
        (bool sent, bytes memory data) = _to.call{value: msg.value}("");
        require(sent, "Failed to send Ether");
    }
}
```

Search for creators or communities

DASHBOARD

WIDGETS TOKEN MANAGE

TOKEN \$SIGJ Change Token

TOKEN HOLDERS 10

AIRDROPS 0

Overview Settings

BRIDGE

Bridge tokens to Polygon Network

GUIDE

LIQUIDITY

Provide Liquidity to your token on Uniswap

GUIDE

Bridge

Deposit

여러분의 기획을 들려주세요!



기술 과제: 토큰을 판매하는 Vendor Account 구현하기

- 토큰의 가격을 정한다. (1 MATIC = 100 Token)
- payable하게 설정된 buyToken() 메소드를 구현한다. ERC20의 오픈제플린 구현체를 참고한다.
- BuyTokens 이벤트를 emit 시켜서 누가 구매자이고, 얼마나 많은 토큰이 구매되었고 그게 몇 MATIC인지 알려준다.
- 지금 내가 만든 토큰을, 배포할 때, Vender Contract에 전부 넘겨준다.
- Vender Contract에 ownership을 넘겨준다.
- Vendor가 토큰을 다시 사들일 수 있도록 한다. 이 과정에서 ERC20에 approve 를 해 주어야 하며, sellTokens 함수를 구현해야 한다.

프론트엔드 코드 및 테스트 코드: https://github.com/Web3-Study-with-Sigrid-Jin/SOPT-web3-study/tree/main/week_2/vendor/

기획 과제: 소셜 토큰 활용하기

- 친구에게 나의 소셜 토큰을 마케팅 또는 판매한다.
- 본인이 기획한 그 코인을 활용처 삼아서 사용한 경험을 공유해주세요.