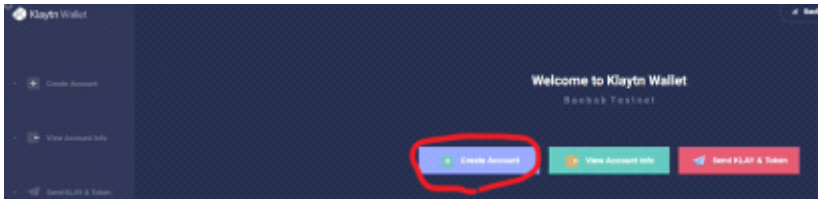
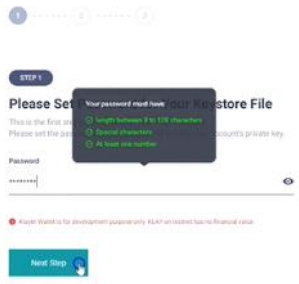
ERC721_Part1

2020년 8월 26일 수요일 오후 2:17

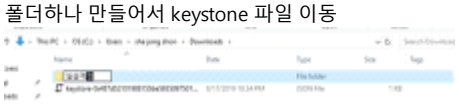
1. 클레이튼 계정3개 만들기
   baobab.wallet.klaytn.com

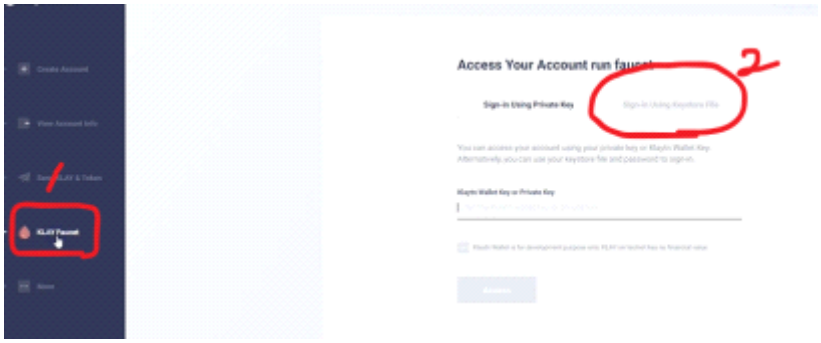


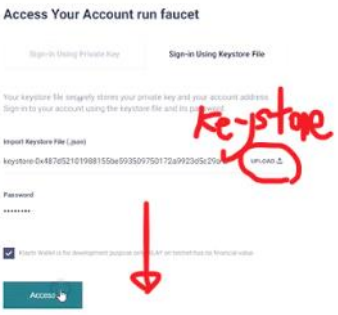


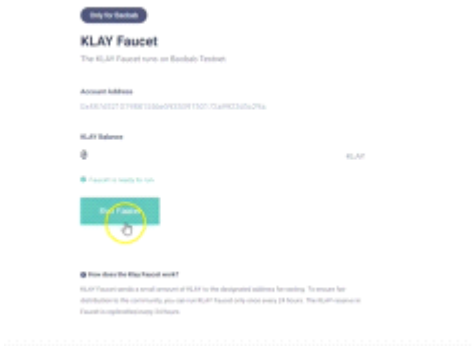
keystone 파일 다운로드 됨

폴더하나 만들어서 keystone 파일 이동









5klay를 받는다.

반복해서 총 3개의 실습용 계정을 만든다
(EX, 계정비번: !@#$%67890, !@#$%11111, !@#$%22222)
==========

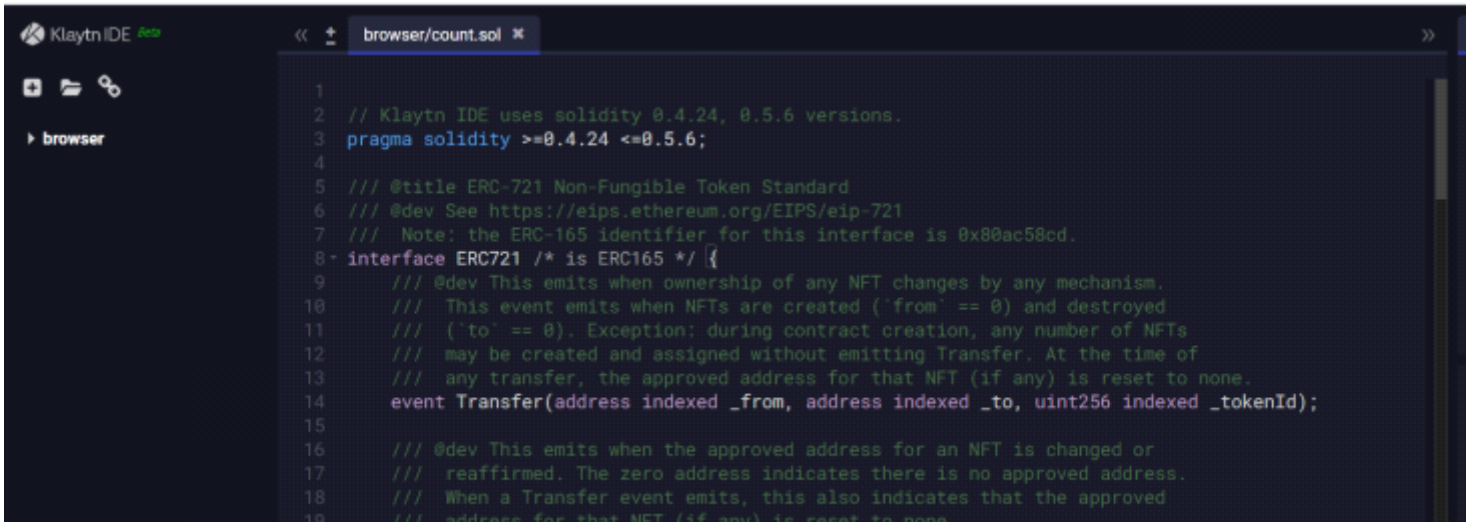
2. EIP 공식 github로 이동.
https://github.com/ethereum/EIPS/blob/master/EIPS/eip-721.md

3. Specification의 interface를 모두 복사

맨 아래쪽에 Interface ERC165는 필요없다.

3. http://ide.klaytn.com/로 이동한다.

4. Interface 붙여넣기

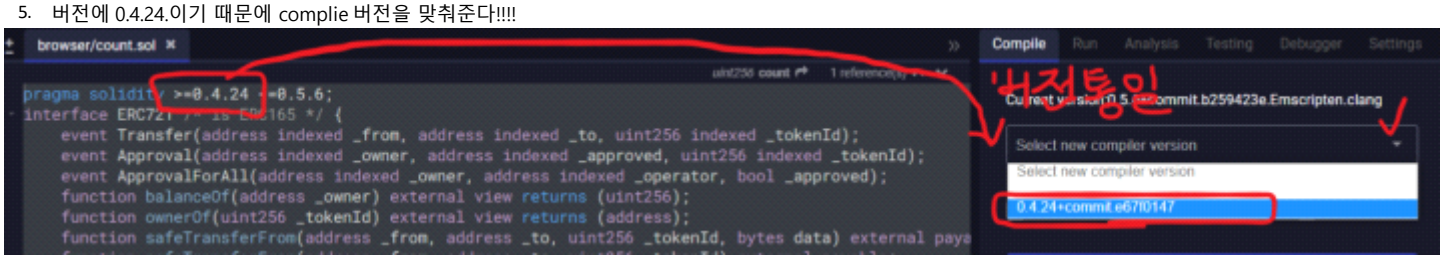


```
    /// @dev This emits when the approved address for an NFT is changed or
    ///  reaffirmed. The zero address indicates there is no approved address.
    ///  When a Transfer event emits, this also indicates that the approved
    ///  address for that NFT (if any) is reset to none.
    event Approval(address indexed _owner, address indexed _approved, uint256 indexed _tokenId);
```

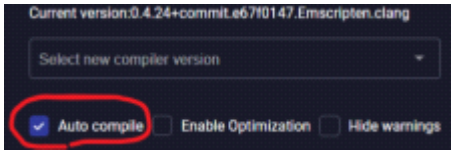
-주석제거버전-

```
pragma solidity >=0.4.24 <=0.5.6;
interface ERC721 /* is ERC165 */ {
  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(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);
}
```

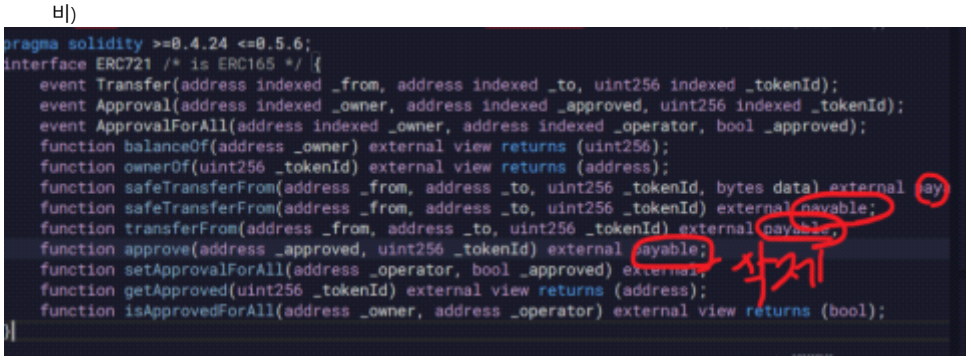
5. 버전에 0.4.24.이기 때문에 complie 버전을 맞춰준다!!!!



6. Auto complie 체크할 것!!!



7. 함수에 payble이 붙어있는데 이게 있으면 돈(klay)를 보내야하기 때문에 지운다!!!(돈낭비)



8. 함수에 external로 public으로 바꾼다
   (※ 가시성을 넓혀서 쓰기 위해. external을 쓰면 컨트렉트 내부 함수끼리 참조해서 쓸 수 없다.)

9. 참고설명

Caveats

The 0.4.20 Solidity interface grammar is not expressive enough to document the ERC-721 standard. A contract which complies with ERC-721 MUST also abide by the following:

- Solidity issue #3412: The above interfaces include explicit mutability guarantees for each function. Mutability guarantees are, in order weak to strong: `payable`, implicit nonpayable, `view`, and `pure`. Your implementation MUST meet the mutability guarantee in this interface and you MAY meet a stronger guarantee. For example, a `payable` function in this interface may be implemented as nonpayble (no state mutability specified) in your contract. We expect a later Solidity release will allow your stricter contract to inherit from this interface, but a workaround for version 0.4.20 is that you can edit this interface to add stricter mutability before inheriting from your contract.
- Solidity issue #3419: A contract that implements `ERC721Metadata` or `ERC721Enumerable` SHALL also implement `ERC721`. ERC-721 implements the requirements of interface ERC-165.
- Solidity issue #2330: If a function is shown in this specification as `external` then a contract will be compliant if it uses `public` visibility. As a workaround for version 0.4.20, you can edit this interface to switch to `public` before inheriting from your contract.
- Solidity issues #3494, #3544: Use of `this.*.selector` is marked as a warning by Solidity, a future version of Solidity will not mark this as an error.

If a newer version of Solidity allows the caveats to be expressed in code, then this EIP MAY be updated and the caveats removed, such will be equivalent to the original specification.

10. 최종 인터페이스(가장 많이 사용되고 있는 버전)

```
pragma solidity >=0.4.24 <=0.5.6;
interface ERC721 /* is ERC165 */ {

  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);
  function ownerOf(uint256 _tokenId) public view returns (address);
  function safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes data)
public;
  function safeTransferFrom(address _from, address _to, uint256 _tokenId) public;
  function transferFrom(address _from, address _to, uint256 _tokenId) public;
  function approve(address _approved, uint256 _tokenId) public;
  function setApprovalForAll(address _operator, bool _approved) public;
  function getApproved(uint256 _tokenId) public view returns (address);
  function isApprovedForAll(address _owner, address _operator) public view returns (bool);
}
```

11. 토큰 발행하기!!
    - 표준규약으로 정해져있지 않다.
    - 그래서 대중적인 토큰 발행함수를 만들 것이다!!!
12. 토큰 발행함수

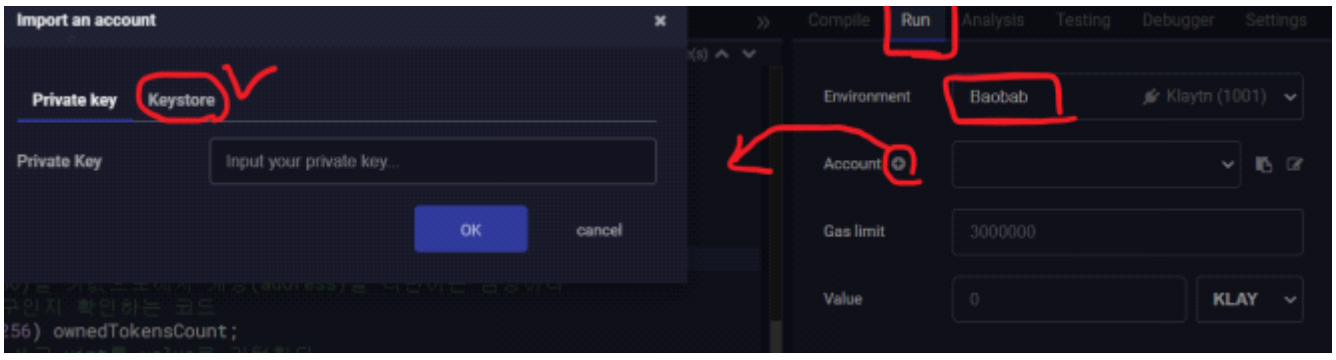
```
pragma solidity >=0.4.24 <=0.5.6;
interface ERC721 /* is ERC165 */ {

  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);
  function ownerOf(uint256 _tokenId) public view returns (address);
  function safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes data) public;
  function safeTransferFrom(address _from, address _to, uint256 _tokenId) public;
  function transferFrom(address _from, address _to, uint256 _tokenId) public;
  function approve(address _approved, uint256 _tokenId) public;
  function setApprovalForAll(address _operator, bool _approved) public;
  function getApproved(uint256 _tokenId) public view returns (address);
  function isApprovedForAll(address _owner, address _operator) public view returns (bool);
}
contract ERC721Implementation is ERC721 {
//is ERC721 = ERC721의 모든 interface를 가져오겠다

    mapping (uint256 => address) tokenOwner;
      //토큰의 아이디(uint256)를 키값으로해서 계정(address)을 리턴하는 맴핑이다
      //즉 토큰의 주인이 누구인지 확인하는 코드
    mapping (address => uint256) ownedTokensCount;
      //address타입을 key로 쓰고 uint를 value로 리턴한다
      //계정주소를 입력하면 해당계정이 몇개의 토큰을 소유하고 있는지 숫자로 리턴
    function mint(address _to, uint _tokenId) public{
      //mint는 발행하다라는 뜻이다
      //매개변수로 계정주소와 토큰아이디를 받겠다!
      //_to는 발행된 토큰을 누가 소유하게 될건지
      //_tokenId는 몇번째 토큰인지(1부터 시작해서 하나씩 증가한다)
      tokenOwner[_tokenId] = _to;
```

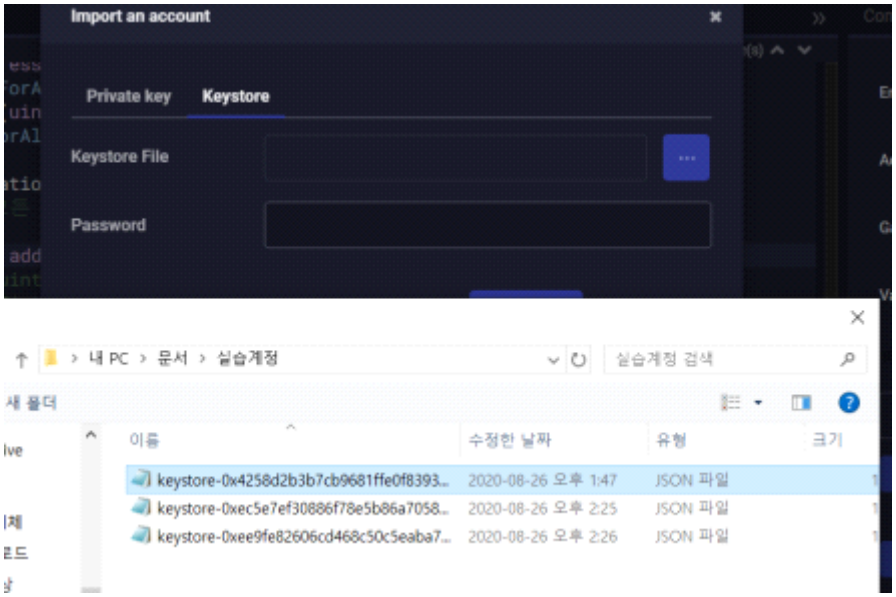
```
        //토큰 아이디의 주인은 _to이다
        ownedTokensCount[_to] += 1;
        //_to계정을 mapping의 key값으로 넘기고 계정이 가지고 있는 토큰 소유개수를 +1 한다.

    }
}
```

13. 계정 발란스 & 토큰 소유자

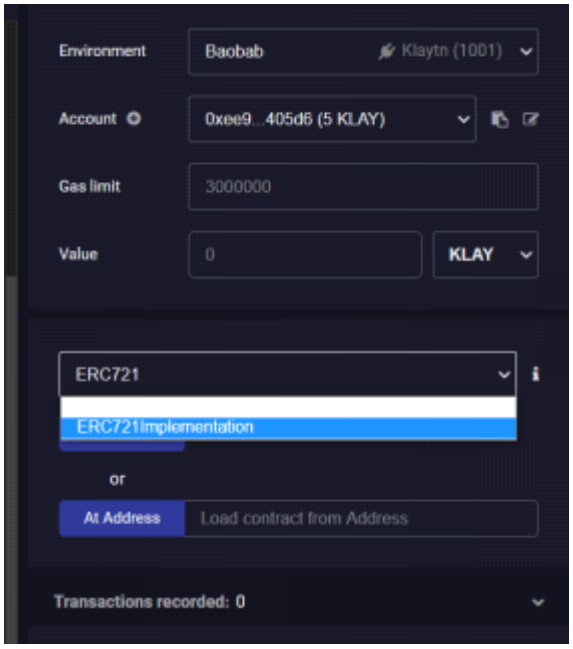
```
pragma solidity >=0.4.24 <=0.5.6;
interface ERC721 /* is ERC165 */ {

    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);
    //매개변수로 _owner 주소를 받고 uint256를 반환한다.
    //해당 _owner 주소가 보유하고 있는 토큰의 갯수를 리턴하는 함수
    //어떤 계정에 몇개의 함수가 있는지 확인
    function ownerOf(uint256 _tokenId) public view returns (address);
    // function safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes data) public;
    // function safeTransferFrom(address _from, address _to, uint256 _tokenId) public;
    // function transferFrom(address _from, address _to, uint256 _tokenId) public;
    // function approve(address _approved, uint256 _tokenId) public;
    // function setApprovalForAll(address _operator, bool _approved) public;
    // function getApproved(uint256 _tokenId) public view returns (address);
    // function isApprovedForAll(address _owner, address _operator) public view returns (bool);
}
contract ERC721Implementation is ERC721 {
//is ERC721 = ERC721의 모든 interface를 가져오겠다

    mapping (uint256 => address) tokenOwner;
      //토큰의 아이디(uint256)를 키값으로해서 계정(address)을 리턴하는 맴핑이다
      //즉 토큰의 주인이 누구인지 확인하는 코드
    mapping (address => uint256) ownedTokensCount;
      //address타입을 key로 쓰고 uint를 value로 리턴한다
      //계정주소를 입력하면 해당계정이 몇개의 토큰을 소유하고 있는지 숫자로 리턴
    function mint(address _to, uint _tokenId) public {
      //mint는 발행하다라는 뜻이다
      //매개변수로 계정주소와 토큰아이디를 받겠다!
      //_to는 발행된 토큰을 누가 소유하게 될건지
      //_tokenId는 몇번째 토큰인지(1부터 시작해서 하나씩 증가한다)
      tokenOwner[_tokenId] = _to;
      //토큰 아이디의 주인은 _to이다
      ownedTokensCount[_to] += 1;
      //_to계정을 mapping의 key값으로 넘기고 계정이 가지고 있는 토큰 소유개수를 +1한다.
    }

    function balanceOf(address _owner) public view returns (uint256){
        return ownedTokensCount[_owner];
        //매개변수 _owner를 매핑의 키값으로 넘기면 오너계정이 소유한 토큰의 갯수를 리턴하게 된다.
    }

    function ownerOf(uint256 _tokenId) public view returns (address) {
        return tokenOwner[_tokenId];
        //토큰의 주인이 누구냐
    }
}
```

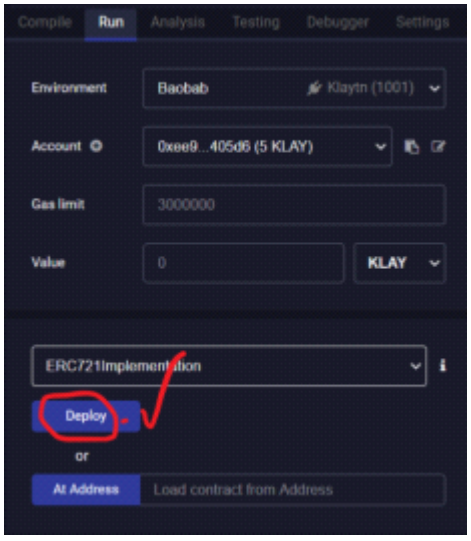
14. test





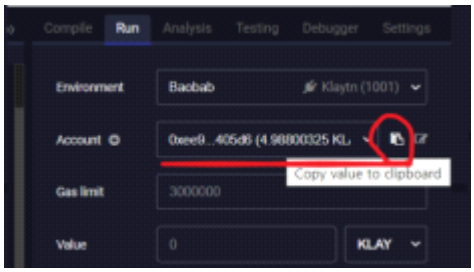
이렇게 3개의 계정 모두 하나씩 불러온다.



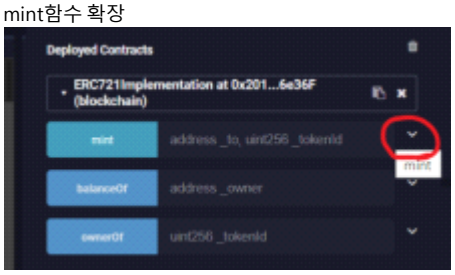


- deploy하면 성공하면서 Deployed contract 생성

- account 주소 복사



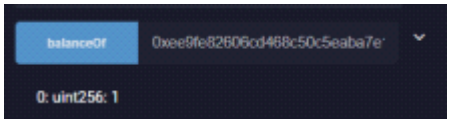
- mint함수 확장



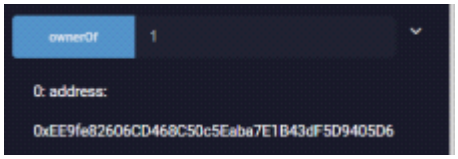
- 주소 붙여넣고 _tokenid는 1입력

- balanceof에 발행에 사용된 계정주소를 입력

- balanceof는 해당주소가소유하고 있는 토큰 개수

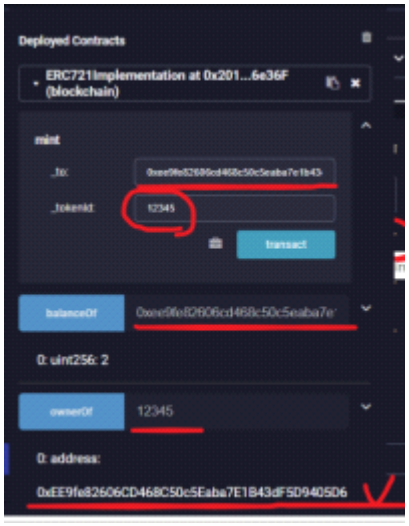


- ownerOf는 토큰아이디를 넘기면 소유계정의 주소가 나온다

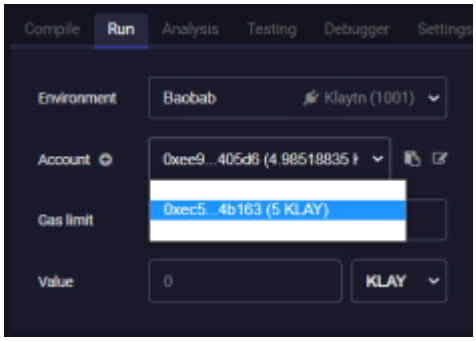


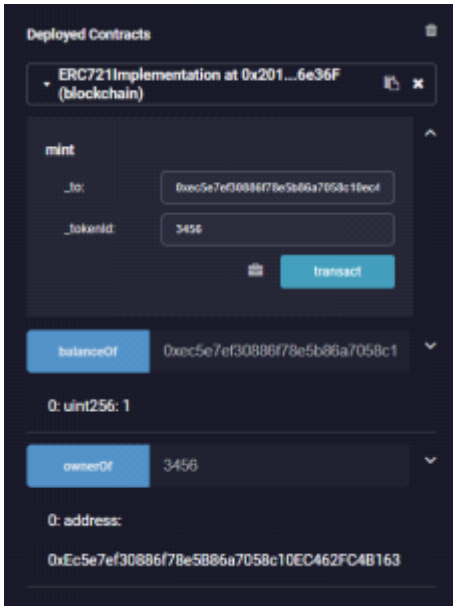
- _tokenID:12345를 mint를 통해 만들면 balanceof로 계정이 소유한 토큰의 갯수를 알수 있으며 ownerOf에 만들어진 tokenID를 넣으면 토큰을 소유한 주소가 나온다.!!!!!!!

-



15. 계정변경해서 테스트





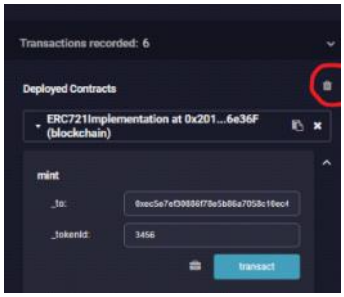
16. 토큰소유권 이전

```
pragma solidity >=0.4.24 <=0.5.6;
interface ERC721 /* is ERC165 */ {

    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);
    //매개변수로 _owner 주소를 받고 uint256를 반환한다.
    //해당 _owner 주소가 보유하고 있는 토큰의 갯수를 리턴하는 함수
    //어떤 계정에 몇개의 함수가 있는지 확인
    function ownerOf(uint256 _tokenId) public view returns (address);
    // function safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes data) public;
    // function safeTransferFrom(address _from, address _to, uint256 _tokenId) public;
    function transferFrom(address _from, address _to, uint256 _tokenId) public;
    // function approve(address _approved, uint256 _tokenId) public;
    // function setApprovalForAll(address _operator, bool _approved) public;
```

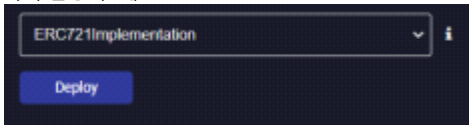
```
    // function getApproved(uint256 _tokenId) public view returns (address);
    // function isApprovedForAll(address _owner, address _operator) public view returns (bool);
}
contract ERC721Implementation is ERC721 {
//is ERC721 = ERC721의 모든 interface를 가져오겠다

    mapping (uint256 => address) tokenOwner;
      //토큰의 아이디(uint256)를 키값으로해서 계정(address)을 리턴하는 맴핑이다
      //즉 토큰의 주인이 누구인지 확인하는 코드
    mapping (address => uint256) ownedTokensCount;
      //address타입을 key로 쓰고 uint를 value로 리턴한다
      //계정주소를 입력하면 해당계정이 몇개의 토큰을 소유하고 있는지 숫자로 리턴
    function mint(address _to, uint _tokenId) public{
      //mint는 발행하다라는 뜻이다
      //매개변수로 계정주소와 토큰아이디를 받겠다!
      // _to는 발행된 토큰을 누가 소유하게 될건지
      // _tokenId는 몇번째 토큰인지(1부터 시작해서 하나씩 증가한다)
      tokenOwner[_tokenId] = _to;
      //토큰 아이디의 주인은 _to이다
      ownedTokensCount[_to] += 1;
      //_to계정을 mapping의 key값으로 넘기고 계정이 가지고 있는 토큰 소유개수를 +1한다.
    }

    function balanceOf(address _owner) public view returns (uint256){
      return ownedTokensCount[_owner];
      //매개변수 _owner를 매핑의 키값으로 넘기면 오너계정이 소유한 토큰의 갯수를 리턴하
게 된다.
    }

    function ownerOf(uint256 _tokenId) public view returns (address) {
      return tokenOwner[_tokenId];
      //토큰의 주인이 누구냐
    }
    function transferFrom(address _from, address _to, uint256 _tokenId) public{
      // from계정에서 to계정으로 옮기겠다.
      address owner = ownerOf(_tokenId);
      require(msg.sender == owner);
      //msg.sender는 함수를 호출한 계정, owner와 같아야 통과하가 된다.
      //유효성 검사
      require(_from != address(0));
      // address(0)은 비어있다는 의미
      require(_to != address(0));
      // from과 to계정이 비어있지 않아야 통과

      ownedTokensCount[_from] -=1;
      //토큰을 from으로 넘기고 1개 차감
      tokenOwner[_tokenId] = address(0);
      //tokenOwner 매핑에 토큰아이디를 넘기고 토큰소유권을 삭제한다.

      ownedTokensCount[_to] += 1;
      tokenOwner[_tokenId] = _to;
      //to계정이 토큰아이디의 새로운 소유자이다.
    }
}
```

17. 테스트

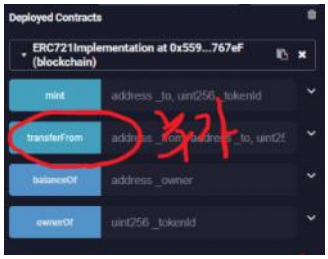
- 기존 transaction들 삭제



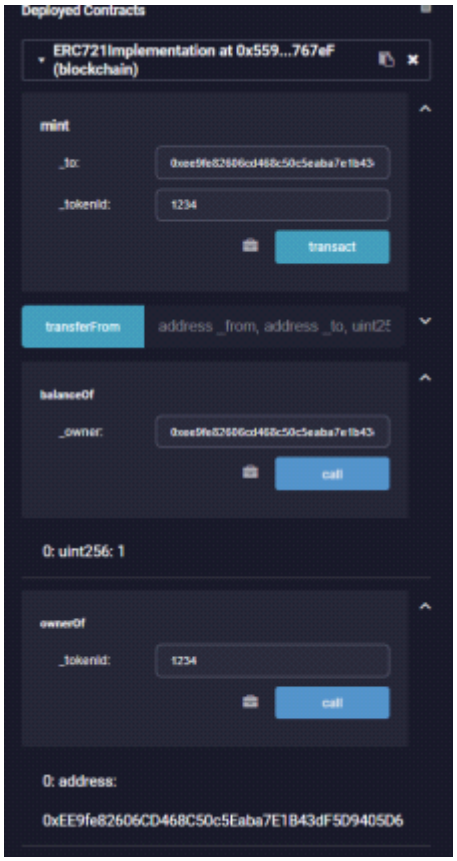
- 다시 설정 후 배포



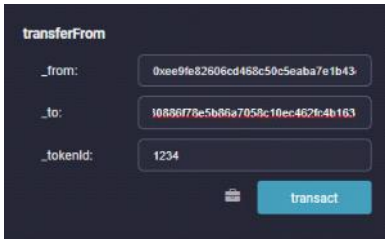
- transferFrom 확인



- 토큰만들어서



- transfer하기

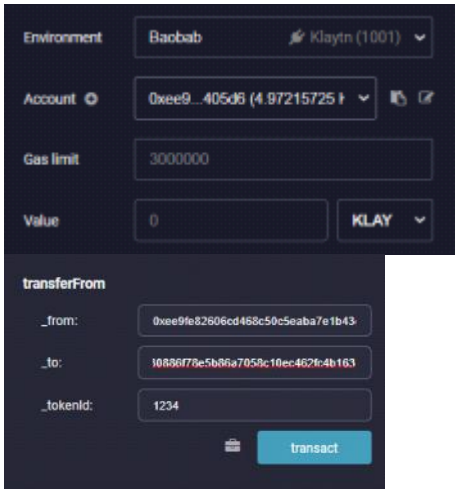


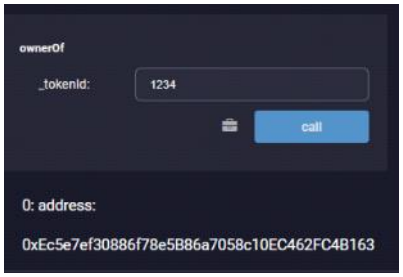
_from은 토큰을 보낼 아이디

_to는 받을 아이디

_tokenid는 보낸 토큰아이디

- transact 버튼을 누르기 전에 다시 account를 토큰을 보낼 아이디로 보내야한다!!!!





---> ownerOf로 확인해보면 소유한 계정의 주소가 바뀌었다!!!!

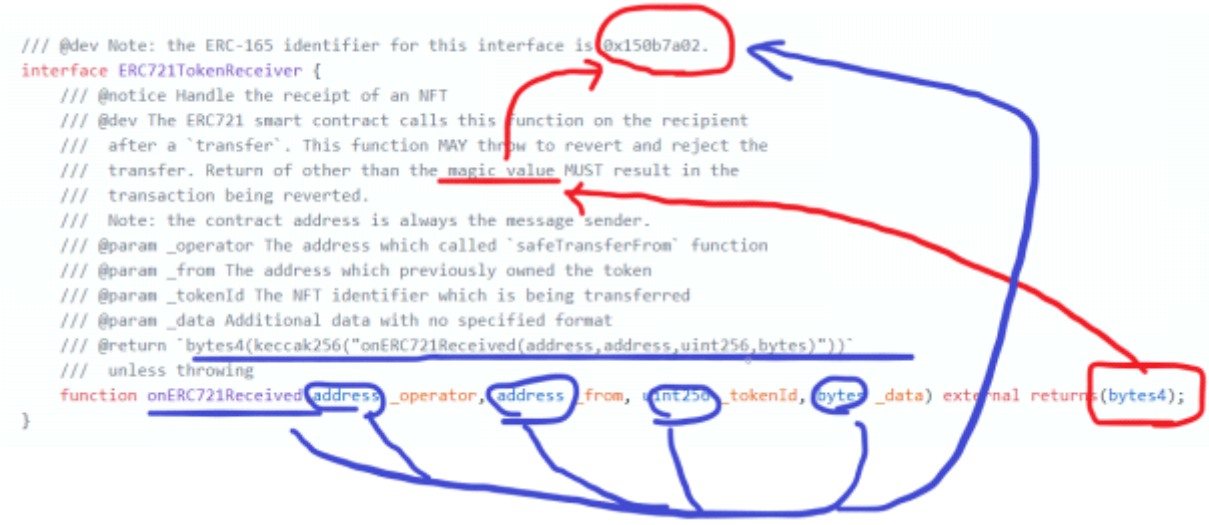
---> 보유한 토큰의 개수도 0으로 감소했다!!!!

---> 받은 계정은 보유 토큰의 수가 1로 증가했다!!!

==========

참고자료: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-721.md

-



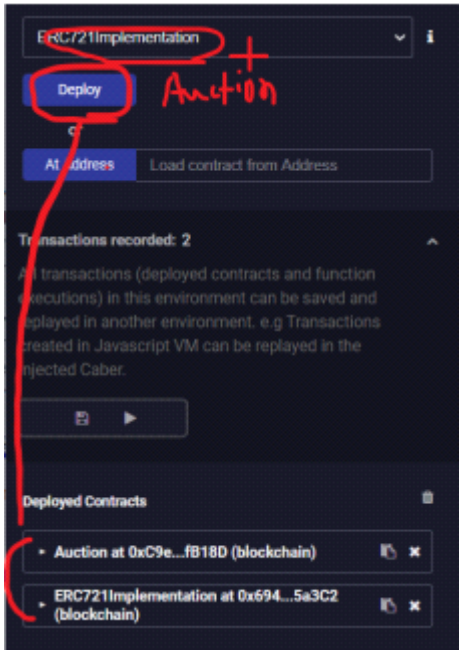
- 코드 설명
- onERC721Received 함수가 파라미터들을 받아서 바이트4타입의 값을 리턴했다면 토큰을 받을 수 있는 컨트렉트이다!
- 리턴된 값은 magic value이다 --> 0x150b7a02 = 함수이름과 매개변수타입(address, address, uint256, bytes)를 암호화시키고 4바이트로 자른 값이다!!!!

18. 토큰 contract 계정(auction)으로 safetransfer

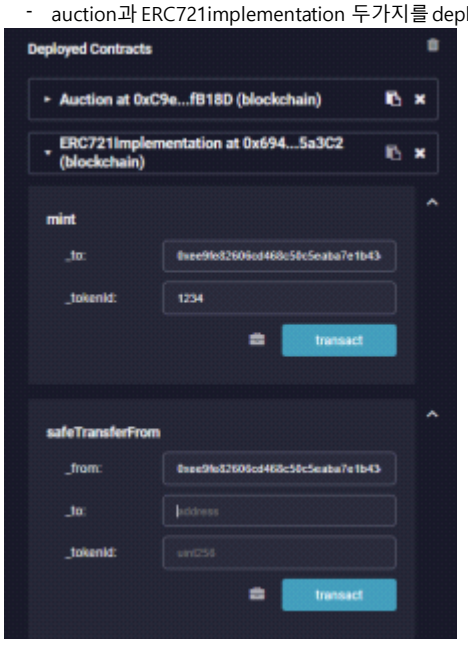
```
pragma solidity >=0.4.24 <=0.5.6;
interface ERC721 /* is ERC165 */ {

    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);
    //매개변수로 _owner 주소를 받고 uint256를 반환한다.
    //해당 _owner 주소가 보유하고 있는 토큰의 갯수를 리턴하는 함수
    //어떤 계정에 몇개의 함수가 있는지 확인
    function ownerOf(uint256 _tokenId) public view returns (address);
    // function safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes data) public;
    function safeTransferFrom(address _from, address _to, uint256 _tokenId) public;
    function transferFrom(address _from, address _to, uint256 _tokenId) public;
```

```
  // function approve(address _approved, uint256 _tokenId) public;
  // function setApprovalForAll(address _operator, bool _approved) public;
  // function getApproved(uint256 _tokenId) public view returns (address);
  // function isApprovedForAll(address _owner, address _operator) public view returns (bool);
}

interface ERC721TokenReceiver {
  function onERC721Received(address _operator, address _from, uint256 _tokenId, bytes _data)
public returns(bytes4);
}//external은 public으로 교환한다!!


contract ERC721Implementation is ERC721 {
//is ERC721 = ERC721의 모든 interface를 가져오겠다

    mapping (uint256 => address) tokenOwner;
      //토큰의 아이디(uint256)를 키값으로해서 계정(address)을 리턴하는 맴핑이다
      //즉 토큰의 주인이 누구인지 확인하는 코드
    mapping (address => uint256) ownedTokensCount;
      //address타입을 key로 쓰고 uint를 value로 리턴한다
      //계정주소를 입력하면 해당계정이 몇개의 토큰을 소유하고 있는지 숫자로 리턴
    function mint(address _to, uint _tokenId) public{
      //mint는 발행하다라는 뜻이다
      //매개변수로 계정주소와 토큰아이디를 받겠다!
      // _to는 발행된 토큰을 누가 소유하게 될건지
      // _tokenId는 몇번째 토큰인지(1부터 시작해서 하나씩 증가한다)
      tokenOwner[_tokenId] = _to;
      //토큰 아이디의 주인은 _to이다
      ownedTokensCount[_to] += 1;
      //_to계정을 mapping의 key값으로 넘기고 계정이 가지고 있는 토큰 소유개수를 +1한다.
    }

    function balanceOf(address _owner) public view returns (uint256){
      return ownedTokensCount[_owner];
      //매개변수 _owner를 매핑의 키값으로 넘기면 오너계정이 소유한 토큰의 갯수를 리턴하
게 된다.
    }

    function ownerOf(uint256 _tokenId) public view returns (address) {
      return tokenOwner[_tokenId];
      //토큰의 주인이 누구냐
    }
    function transferFrom(address _from, address _to, uint256 _tokenId) public{
      // from계정에서 to계정으로 옮기겠다.
      address owner = ownerOf(_tokenId);
      require(msg.sender == owner);
      //msg.sender는 함수를 호출한 계정, owner와 같아야 통과가 된다.(중요!!!!!)
      //유효성 검사
      require(_from != address(0));
      // address(0)은 비어있다는 의미
      require(_to != address(0));
      // from과 to계정이 비어있지 않아야 통과

      ownedTokensCount[_from] -=1;
      //토큰을 from으로 넘기고 1개 차감
      tokenOwner[_tokenId] = address(0);
      //tokenOwner 매핑에 토큰아이디를 넘기고 토큰소유권을 삭제한다.

      ownedTokensCount[_to] += 1;
      tokenOwner[_tokenId] = _to;
      //to계정이 토큰아이디의 새로운 소유자이다.
    }

    function safeTransferFrom(address _from, address _to, uint256 _tokenId) public{
      transferFrom(_from, _to, _tokenId);
      if (isContract(_to)){
        bytes4 returnValue = ERC721TokenReceiver(_to).onERC721Received(msg.sender, _from,
_tokenId, '');
        //컨트렉트계정인 (_to)주소에 ERC721TokenReceiver 인터페이스가 존재한다.
        // 그 안에 있는 onERC721Received 함수에 인자들을 넘겨서 호출한다.
        // 함수가 구현되어 있으면 magic value를 리턴하며 returnValue에 저장한다
        require(returnValue == 0x150b7a02);
        // returnValue의 값이 magicValue여야 통과가 가능하다!!!!
      }
    }

    function safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes data) public{
```

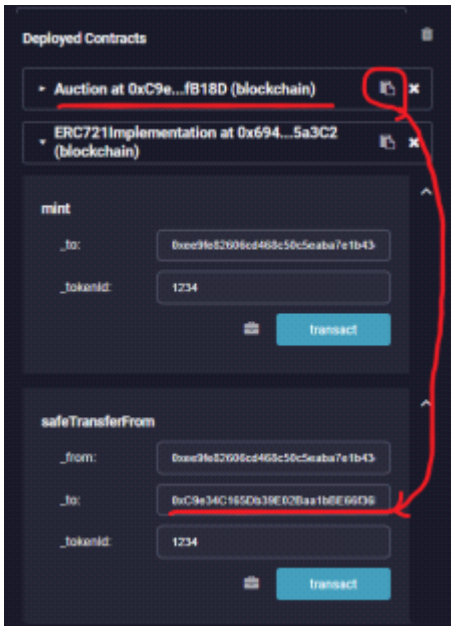
```
            transferFrom(_from, _to, _tokenId);
        if (isContract(_to)){
            bytes4 returnValue = ERC721TokenReceiver(_to).onERC721Received(msg.sender, _from,
_tokenId, data);
            //위의 safeTransferFrom와 같지만 bytes data 매개변수가 추가된다.
            // 프런트엔드쪽에서 만들 변수를 미리 지정해놓는다
            require(returnValue == 0x150b7a02);
        }
    }
    function isContract(address _addr) private view returns (bool){
        //address가 contract계정이면 true 아니면 false로 리턴하는 함수
        uint256 size;
        assembly { size:= extcodesize(_addr)}
        return size >0;
        // extcodesize에 주소를 넣었을 때 0이면 일반계정이고 0보다 크면 컨트렉트계정

    }
}
contract Auction is ERC721TokenReceiver{
    //ERC721TokenReceiver 상속받음
    function onERC721Received(address _operator, address _from, uint256 _tokenId, bytes _data)
public returns(bytes4){
        return bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"));
    // safeTransferFrom 에서 if 조건과 require 조건을 contract Auction이 모두 충족하고 있다!!유
효성검사 통과가능!
    // token 이동 기능이 없다(테스트용이기 때문)

    }
}
```



- auction과 ERC721implementation 두가지를 deploy 한다.



- mint로 한계정에 tokenid 1234를 생성한다



- safeTransferFrom에서 _from에 통큰보유 계정을 넣고 _to에 auction contract의 주소를 복사해서 넣고 transact!!!

- ownerOf에서 tokenId를 검색하면 auction contract의 주소가 나온다!!

==========

19. 토큰 승인(제3자기 내 토큰을 대신 처리해준다)

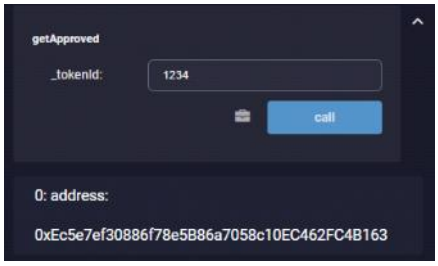
```
pragma solidity >=0.4.24 <=0.5.6;
interface ERC721 /* is ERC165 */ {

  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);
  //매개변수로 _owner 주소를 받고 uint256를 반환한다.
  //해당 _owner 주소가 보유하고 있는 토큰의 갯수를 리턴하는 함수
  //어떤 계정에 몇개의 함수가 있는지 확인
  function ownerOf(uint256 _tokenId) public view returns (address);
  function safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes data) public;
  function safeTransferFrom(address _from, address _to, uint256 _tokenId) public;
  function transferFrom(address _from, address _to, uint256 _tokenId) public;
  function approve(address _approved, uint256 _tokenId) public;
  // function setApprovalForAll(address _operator, bool _approved) public;
  function getApproved(uint256 _tokenId) public view returns (address);
  // function isApprovedForAll(address _owner, address _operator) public view returns (bool);
}

interface ERC721TokenReceiver {
  function onERC721Received(address _operator, address _from, uint256 _tokenId, bytes _data)
public returns(bytes4);
}//external은 public으로 교환한다!!


contract ERC721Implementation is ERC721 {
//is ERC721 = ERC721의 모든 interface를 가져오겠다

    mapping (uint256 => address) tokenOwner;
      //토큰의 아이디(uint256)를 키값으로해서 계정(address)을 리턴하는 맴핑이다
      //즉 토큰의 주인이 누구인지 확인하는 코드
    mapping (address => uint256) ownedTokensCount;
      //address타입을 key로 쓰고 uint를 value로 리턴한다
      //계정주소를 입력하면 해당계정이 몇개의 토큰을 소유하고 있는지 숫자로 리턴
    mapping (uint256 => address) tokenApprovals;
```

```
    //권한을 갖게되는 계정을 저장해야한다(appover에 사용)
    //토큰 아이디를 키값으로해서 계정주소를 저장한다

  function mint(address _to, uint _tokenId) public{
    //mint는 발행하다라는 뜻이다
    //매개변수로 계정주소와 토큰아이디를 받겠다!
    //_to는 발행된 토큰을 누가 소유하게 될건지
    //_tokenId는 몇번째 토큰인지(1부터 시작해서 하나씩 증가한다)
    tokenOwner[_tokenId] = _to;
    //토큰 아이디의 주인은 _to이다
    ownedTokensCount[_to] += 1;
    //_to계정을 mapping의 key값으로 넘기고 계정이 가지고 있는 토큰 소유개수를 +1한다.
  }

  function balanceOf(address _owner) public view returns (uint256){
    return ownedTokensCount[_owner];
    //매개변수 _owner를 매핑의 키값으로 넘기면 오너계정이 소유한 토큰의 갯수를 리턴하
게 된다.
  }

  function ownerOf(uint256 _tokenId) public view returns (address) {
    return tokenOwner[_tokenId];
    //토큰의 주인이 누구냐
  }
  function transferFrom(address _from, address _to, uint256 _tokenId) public{
    // from계정에서 to계정으로 옮기겠다.
    address owner = ownerOf(_tokenId);
    require(msg.sender == owner || getApproved(_tokenId) == msg.sender);
    //msg.sender는 함수를 호출한 계정, owner와 같아야 통과가 된다.(중요!!!!!)
    //getApproved에서 리턴하는 계정이 함수를 호출한 계정과 동일하다면 통과시킨다
    //유효성 검사
    require(_from != address(0));
    // address(0)은 비어있다는 의미
    require(_to != address(0));
    // from과 to계정이 비어있지 않아야 통과

    ownedTokensCount[_from] -=1;
    //토큰을 from으로 넘기고 1개 차감
    tokenOwner[_tokenId] = address(0);
    //tokenOwner 매핑에 토큰아이디를 넘기고 토큰소유권을 삭제한다.

    ownedTokensCount[_to] += 1;
    tokenOwner[_tokenId] = _to;
    //to계정이 토큰아이디의 새로운 소유자이다.
  }

  function safeTransferFrom(address _from, address _to, uint256 _tokenId) public{
    transferFrom(_from, _to, _tokenId);
    if (isContract(_to)){
      bytes4 returnValue = ERC721TokenReceiver(_to).onERC721Received(msg.sender, _from,
_tokenId, '');
      //컨트렉트계정인 (_to)주소에 ERC721TokenReceiver 인터페이스가 존재한다.
      // 그 안에 있는 onERC721Received 함수에 인자들을 넘겨서 호출한다.
      // 함수가 구현되어 있으면 magic value를 리턴하며 returnValue에 저장한다
      require(returnValue == 0x150b7a02);
      // returnValue의 값이 magicValue여야 통과가 가능하다!!!!
    }
  }

  function safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes data) public{
      transferFrom(_from, _to, _tokenId);
    if (isContract(_to)){
      bytes4 returnValue = ERC721TokenReceiver(_to).onERC721Received(msg.sender, _from,
_tokenId, data);
      //위의 safeTransferFrom와 같지만 bytes data 매개변수가 추가된다.
      // 프런트엔드쪽에서 만들 변수를 미리 지정해놓는다
      require(returnValue == 0x150b7a02);
    }
  }
  function isContract(address _addr) private view returns (bool){
    //address가 contract계정이면 true 아니면 false로 리턴하는 함수
    uint256 size;
    assembly { size := extcodesize(_addr)}
    return size > 0;
    // extcodesize에 주소를 넣었을 때 0이면 일반계정이고 0보다 크면 컨트렉트계정

  `
```

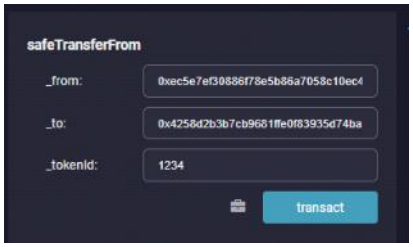
```
    }

    function approve(address _approved, uint256 _tokenId) public {
    // 이 함수는 권한을 받게되는 계정과 토큰아이디를 매개변수로 받는다
        address owner = ownerOf(_tokenId);
        // 토큰 소유자 계정을 불러온다
        require(_approved != owner);
        // 원한을 받게되는 계정이 소유자 계정이 아니여야 통과시킨다
        require(msg.sender == owner);
        //approve를 호출한 계정(msg.sender)이 소유자 계정이어야 한다.
        tokenApprovals[_tokenId] = _approved;
    }

    function getApproved(uint256 _tokenId) public view returns (address) {
        return tokenApprovals[_tokenId];
    }
}

contract Auction is ERC721TokenReceiver {
    //ERC721TokenReceiver 상속받음
    function onERC721Received(address _operator, address _from, uint256 _tokenId, bytes _data)
public returns(bytes4){
        return bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"));
    // safeTransferFrom 에서 if 조건과 require 조건을 contract Auction이 모두 충족하고 있다!!유
효성검사 통과가능!
    // token 이동 기능이 없다(테스트용이ㅣ기 때문)
    }
}
```

- 1번 계정으로 token생성

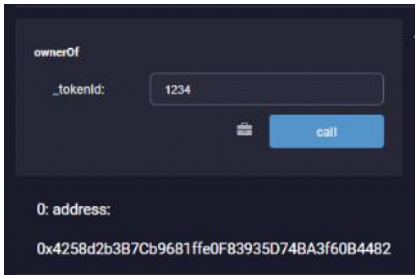
- 2번 계정에 토큰 권한 승인!

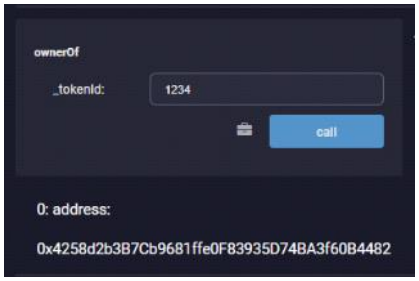


- 권한받은 계정 확인 == 2번계정



- 2번 계정이 1번 계정을 대신해서 3번계정에게 토큰을 전달





- 토큰1234의 주인은 3번 계정이 된다!!!

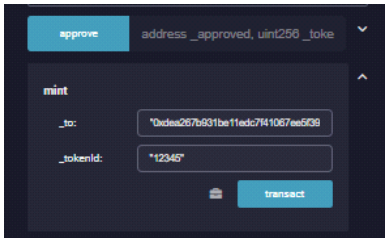
==========

20. 토큰 전체 승인

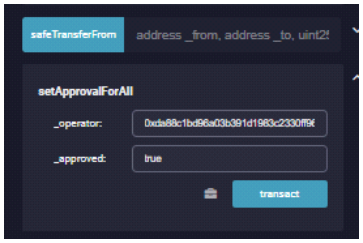
```
pragma solidity >=0.4.24 <=0.5.6;
interface ERC721 /* is ERC165 */ {

  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);
  //매개변수로 _owner 주소를 받고 uint256를 반환한다.
  //해당 _owner 주소가 보유하고 있는 토큰의 갯수를 리턴하는 함수
  //어떤 계정에 몇개의 함수가 있는지 확인
  function ownerOf(uint256 _tokenId) public view returns (address);
  function safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes data) public;
  function safeTransferFrom(address _from, address _to, uint256 _tokenId) public;
  function transferFrom(address _from, address _to, uint256 _tokenId) public;
  function approve(address _approved, uint256 _tokenId) public;
  function setApprovalForAll(address _operator, bool _approved) public;
  //계정이 소유한 모든 토큰을 전송할 수 있는 권한은 부여한다
  function getApproved(uint256 _tokenId) public view returns (address);
  function isApprovedForAll(address _owner, address _operator) public view returns (bool);
  //

}

interface ERC721TokenReceiver {
  function onERC721Received(address _operator, address _from, uint256 _tokenId, bytes _data)
public returns(bytes4);
}//external은 public으로 교환한다!!


contract ERC721Implementation is ERC721 {
//is ERC721 = ERC721의 모든 interface를 가져오겠다

    mapping (uint256 => address) tokenOwner;
      //토큰의 아이디(uint256)를 키값으로해서 계정(address)을 리턴하는 맴핑이다
      //즉 토큰의 주인이 누구인지 확인하는 코드
    mapping (address => uint256) ownedTokensCount;
      //address타입을 key로 쓰고 uint를 value로 리턴한다
      //계정주소를 입력하면 해당계정이 몇개의 토큰을 소유하고 있는지 숫자로 리턴
    mapping (uint256 => address) tokenApprovals;
      //권한을 갖게되는 계정을 저장해야한다(appover에 사용)
      //토큰 아이디를 키값으로해서 계정주소를 저장한다
    mapping (address ==> mapping(address => bool)) operatorApprovals;
      // 주소형을 키값으로해서 value 값으로 또 mapping(주소형을 키값을 true/false 리턴)이 들
어간다
      // 누가 누구에게 권한 부여를 했는가


    function mint(address _to, uint _tokenId) public{
      //mint는 발행하다라는 뜻이다
      //매개변수로 계정주소와 토큰아이디를 받겠다!
      // _to는 발행된 토큰을 누가 소유하게 될건지
      // _tokenId는 몇번째 토큰인지(1부터 시작해서 하나씩 증가한다)
      tokenOwner[_tokenId] = _to;
      //토큰 아이디의 주인은 _to이다
      ownedTokensCount[_to] += 1;
      //_to계정을 mapping의 key값으로 넘기고 계정이 가지고 있는 토큰 소유개수를 +1한다.
    }

    function balanceOf(address _owner) public view returns (uint256){
      return ownedTokensCount[_owner];
      //매개변수 _owner를 매핑의 키값으로 넘기면 오너계정이 소유한 토큰의 갯수를 리턴하
```

게 된다.
```
    }

    function ownerOf(uint256 _tokenId) public view returns (address) {
        return tokenOwner[_tokenId];
        //토큰의 주인이 누구냐
    }
    function transferFrom(address _from, address _to, uint256 _tokenId) public{
        // from계정에서 to계정으로 옮기겠다.
        address owner = ownerOf(_tokenId);
        require(msg.sender == owner || getApproved(_tokenId) == msg.sender ||
isApprovedForAll(owner, msg.sender));
        //msg.sender는 함수를 호출한 계정, owner와 같아야 통과가 된다.(중요!!!!!)
        //getApproved에서 리턴하는 계정이 함수를 호출한 계정과 동일하다면 통과시킨다
        //owner계정과 함수호출 계정을 넘겨서 msg.sender가 owner토큰의 전송 권한이 있는지
확인(true.false)
        //소유자의 전체 토큰의 전송 권한을 가진 계정이 호출했을 때 토큰 전송 가능.
        //유효성 검사
        require(_from != address(0));
        // address(0)은 비어있다는 의미
        require(_to != address(0));
        // from과 to계정이 비어있지 않아야 통과

        ownedTokensCount[_from] -=1;
        //토큰을 from으로 넘기고 1개 차감
        tokenOwner[_tokenId] = address(0);
        //tokenOwner 매핑에 토큰아이디를 넘기고 토큰소유권을 삭제한다.

        ownedTokensCount[_to] += 1;
        tokenOwner[_tokenId] = _to;
        //to계정이 토큰아이디의 새로운 소유자이다.
    }

    function safeTransferFrom(address _from, address _to, uint256 _tokenId) public{
        transferFrom(_from, _to, _tokenId);
        if (isContract(_to)){
            bytes4 returnValue = ERC721TokenReceiver(_to).onERC721Received(msg.sender, _from,
_tokenId, '');
            //컨트렉트계정인 (_to)주소에 ERC721TokenReceiver 인터페이스가 존재한다.
            // 그 안에 있는 onERC721Received 함수에 인자들을 넘겨서 호출한다.
            // 함수가 구현되어 있으면 magic value를 리턴하며 returnValue에 저장한다
            require(returnValue == 0x150b7a02);
            // returnValue의 값이 magicValue여야 통과가 가능하다!!!!
        }
    }

    function safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes data) public{
        transferFrom(_from, _to, _tokenId);
        if (isContract(_to)){
            bytes4 returnValue = ERC721TokenReceiver(_to).onERC721Received(msg.sender, _from,
_tokenId, data);
            //위의 safeTransferFrom와 같지만 bytes data 매개변수가 추가된다.
            // 프런트엔드쪽에서 만들 변수를 미리 지정해놓는다
            require(returnValue == 0x150b7a02);
        }
    }

    function approve(address _approved, uint256 _tokenId) public{
    // 이 함수는 권한을 받게되는 계정과 토큰아이디를 매개변수로 받는다
        address owner = ownerOf(_tokenId);
        // 토큰 소유자 계정을 불러온다
        require(_approved != owner);
        // 원한을 받게되는 계정이 소유자 계정이 아니여야 통과시킨다
        require(msg.sender == owner);
        //approve를 호출한 계정(msg.sender)이 소유자 계정이어야 한다.
        tokenApprovals[_tokenId] = _approved;
    }

    function getApproved(uint256 _tokenId) public view returns (address){
        return tokenApprovals[_tokenId];
    }

    function setApprovalForAll(address _operator, bool _approved) public{
        // _operator는 모든 토큰을 대신 운용해줄 계정
        // _operator는 일반계정 contract 계정 둘다가능
        // 권한을 부여할지를 approve를 이이동
        require (_operator != msg.sender);
```

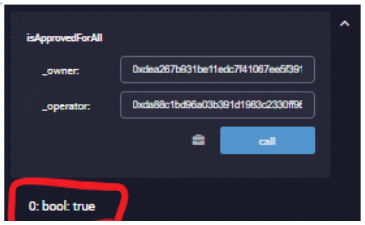
```
    // _operator 계정이 현재 함수를 호출한 계정이 되어서는 안된다.
    operatorApprovals[msg.sender][_operator] = _approved;
    //함수를 호출한 계정이 오퍼레이터 계정에게 권한을 부여할 것인지 말것인지를
_approved 인자를 통해 mapping에 저장
    // _approved가 true면 권한 부여, false면 권한 취소.
  }

  function isApprovedForAll(address _owner, address _operator) public view returns (bool){
    // _owner와 _operator를 받고 bool을 리리턴하는
    // 토큰의 오너가 _operator에게 권한을 부여했는지 안했는지 확인하는 함수
    returns operatorApprovals[_owner][_operator];
    // 토큰 소유자 계정을 키값으로 해서 _operator계정을 확인해서 ower가 _operator에게
권한을 줬는지 안줬는지 true or false를 리턴하게 한다.
  }

  function isContract(address _addr) private view returns (bool){
    //address가 contract계정이면 true 아니면 false로 리턴하는 함수
    uint256 size;
    assembly { size := extcodesize(_addr)}
    return size >0;
    // extcodesize에 주소를 넣었을 때 0이면 일반계정이고 0보다 크면 컨트렉트계정

  }
}

contract Auction is ERC721TokenReceiver {
   //ERC721TokenReceiver 상속받음
  function onERC721Received(address _operator, address _from, uint256 _tokenId, bytes _data)
public returns(bytes4){
    return bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"));
  // safeTransferFrom 에서 if 조건과 require 조건을 contract Auction이 모두 충족하고 있다!!유
효성검사 통과가능!
  // token 이동 기능이 없다(테스트용이 | 기 때문)
  }
}
```



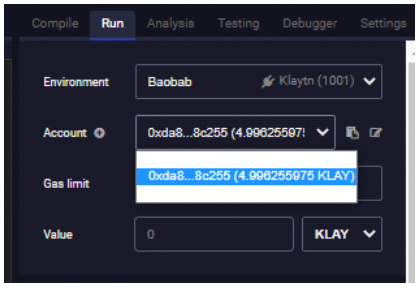
- 1번 계정 토큰 생성



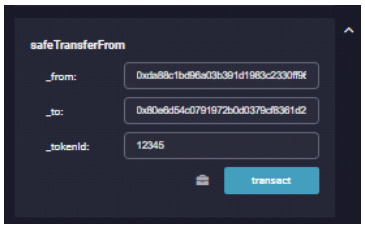
- 2번 계정에 토큰 전송 권한 true로 설정



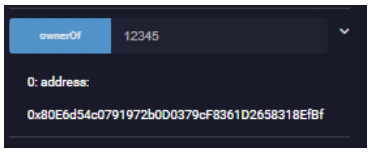
- 1번 계정이 2번 계정에게 모든 권한을 주었는지 true로 확인!



- ***권한을 부여받은 2번 계정을 이동*****



- 2번 계정으로 이동해서 3번계정에게 처음 만든 12345토큰을 전송



- 12345 토큰의 소유를 확인하면 3번 계정으로 나온다!!!!!!!

==========

21. ERC165의 기능

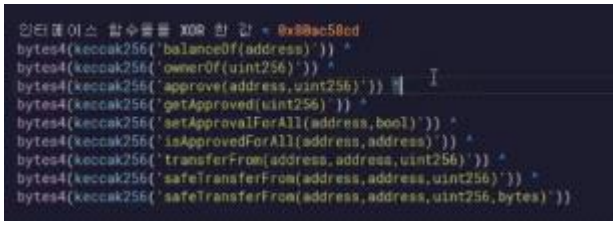
```
/// Note: the ERC-165 identifier for this interface is 0x80ac58cd.

interface ERC165 {
    /// @notice Query if a contract implements an interface
    /// @param interfaceID The interface identifier, as specified in ERC-165
    /// @dev Interface identification is specified in ERC-165. This function
    ///  uses less than 30,000 gas.
    /// @return `true` if the contract implements `interfaceID` and
    ///  `interfaceID` is not 0xffffffff, `false` otherwise
    function supportsInterface(bytes4 interfaceID) external view returns (bool);
}
```

출처: <https://github.com/ethereum/EIPs/blob/master/EIPS/eip-721.md>

// contract의 어떤 interface를 상속받고 있는지 확인

// 인터페이스 ID를 넘기면 boolen을 리턴한다

// 인터페이스 ID는 식별자(ERC721 = 0x80ac58cd)

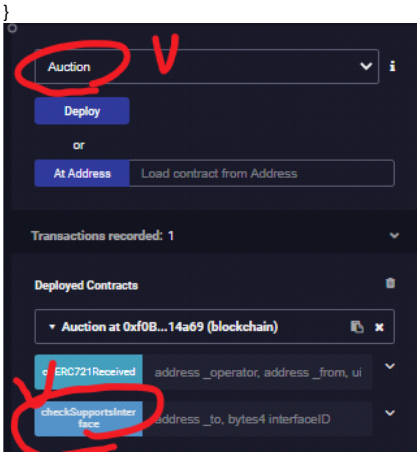



```
pragma solidity >=0.4.24 <=0.5.6;
interface ERC721 /* is ERC165 */ {
    /// Note: the ERC-165 identifier for this interface is 0x80ac58cd.

    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);
    //매개변수로 _owner 주소를 받고 uint256를 반환한다.
    //해당 _owner 주소가 보유하고 있는 토큰의 갯수를 리턴하는 함수
    //어떤 계정에 몇개의 함수가 있는지 확인
    function ownerOf(uint256 _tokenId) public view returns (address);
    function safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes data) public;
    function safeTransferFrom(address _from, address _to, uint256 _tokenId) public;
    function transferFrom(address _from, address _to, uint256 _tokenId) public;
    function approve(address _approved, uint256 _tokenId) public;
    function setApprovalForAll(address _operator, bool _approved) public;
    //계정이 소유한 모든 토큰을 전송할 수 있는 권한은 부여한다
    function getApproved(uint256 _tokenId) public view returns (address);
    function isApprovedForAll(address _owner, address _operator) public view returns (bool);


}


interface ERC721TokenReceiver{
    function onERC721Received(address _operator, address _from, uint256 _tokenId, bytes _data)
public returns(bytes4);
}//external은 public으로 교환한다!!
```

```
interface ERC165 {
    function supportsInterface(bytes4 interfaceID) public view returns (bool);
    // contract의 어떤 interface를 상속받고 있는지 확인
    // 인터페이스 ID를 넘기면 boolen을 리턴한다
    // 인터페이스 ID는 식별자(ERC721 = 0x80ac58cd )
}

contract ERC721Implementation is ERC721 {
//is ERC721 = ERC721의 모든 interface를 가져오겠다

    mapping (uint256 => address) tokenOwner;
      //토큰의 아이디(uint256)를 키값으로해서 계정(address)을 리턴하는 맴핑이다
      //즉 토큰의 주인이 누구인지 확인하는 코드
    mapping (address => uint256) ownedTokensCount;
      //address타입을 key로 쓰고 uint를 value로 리턴한다
      //계정주소를 입력하면 해당계정이 몇개의 토큰을 소유하고 있는지 숫자로 리턴
    mapping (uint256 => address) tokenApprovals;
      //권한을 갖게되는 계정을 저장해야한다(appover에 사용)
      //토큰 아이디를 키값으로해서 계정주소를 저장한다
    mapping (address => mapping(address => bool)) operatorApprovals;
      // 주소형을 키값으로해서 value 값으로 또 mapping(주소형을 키값을 true/false 리턴)이 들
어간다
      // 누가 누구에게 권한 부여를 했는가
    mapping (bytes4 => bool) supportsInterfaces;
      // bytes4는 인터페이스 식별자를 뜻한다
      // 외부에서 mapping을 통해 우리가 쓰고 있는 ERC721Implementation contract이 특정 인
터페이스를 쓰는지 안쓰는지 확인할 수 있게 한다.
      // 이값의 저장은 생성자에서 초기화시키면 된다.

    constructor() public{
        supportsInterfaces[0x80ac58cd] = true;
        // 이 컨트렉은 ERC721을 상속받고 함수들을 구현하고 있다고 세팅함

    }
    function mint(address _to, uint _tokenId) public{
        //mint는 발행하다라는 뜻이다
        //매개변수로 계정주소와 토큰아이디를 받겠다!
        // _to는 발행된 토큰을 누가 소유하게 될건지
        // _tokenId는 몇번째 토큰인지(1부터 시작해서 하나씩 증가한다)
        tokenOwner[_tokenId] = _to;
        //토큰 아이디의 주인은 _to이다
        ownedTokensCount[_to] += 1;
        //_to계정을 mapping의 key값으로 넘기고 계정이 가지고 있는 토큰 소유개수를 +1한다.
    }

    function balanceOf(address _owner) public view returns (uint256){
        return ownedTokensCount[_owner];
        //매개변수 _owner를 매핑의 키값으로 넘기면 오너계정이 소유한 토큰의 갯수를 리턴하
게 된다.
    }

    function ownerOf(uint256 _tokenId) public view returns (address) {
        return tokenOwner[_tokenId];
        //토큰의 주인이 누구냐
    }
    function transferFrom(address _from, address _to, uint256 _tokenId) public{
        // from계정에서 to계정으로 옮기겠다.
        address owner = ownerOf(_tokenId);
        require(msg.sender == owner || getApproved(_tokenId) == msg.sender ||
isApprovedForAll(owner, msg.sender) == true);
        //msg.sender는 함수를 호출한 계정, owner와 같아야 통과가 된다.(중요!!!!!)
        //getApproved에서 리턴하는 계정이 함수를 호출한 계정과 동일하다면 통과시킨다
        //owner계정과 함수호출 계정을 넘겨서 msg.sender가 owner토큰의 전송 권한이 있는지
확인(true.false)
        //소유자의 전체 토큰의 전송 권한을 가진 계정이 호출했을 때 토큰 전송 가능.
        //유효성 검사
        require(_from != address(0));
        // address(0)은 비어있다는 의미
        require(_to != address(0));
        // from과 to계정이 비어있지 않아야 통과

        ownedTokensCount[_from] -= 1;
```

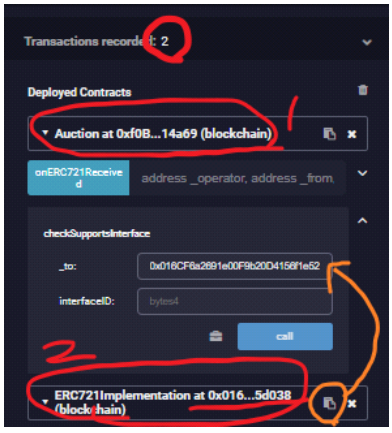
```
    //토큰을 from으로 넘기고 1개 차감
    tokenOwner[_tokenId] = address(0);
    //tokenOwner 매핑에 토큰아이디를 넘기고 토큰소유권을 삭제한다.

    ownedTokensCount[_to] += 1;
    tokenOwner[_tokenId] = _to;
    //to계정이 토큰아이디의 새로운 소유자이다.
  }

  function safeTransferFrom(address _from, address _to, uint256 _tokenId) public{
    transferFrom(_from, _to, _tokenId);
    if (isContract(_to)){
      bytes4 returnValue = ERC721TokenReceiver(_to).onERC721Received(msg.sender, _from,
_tokenId, '');
      //컨트렉트계정인 (_to)주소에 ERC721TokenReceiver 인터페이스가 존재한다.
      // 그 안에 있는 onERC721Received 함수에 인자들을 넘겨서 호출한다.
      // 함수가 구현되어 있으면 magic value 를 리턴하며 returnValue 에 저장한다
      require(returnValue == 0x150b7a02);
      // returnValue 의 값이 magicValue 여야 통과가 가능하다!!!!
    }
  }

  function safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes data) public{
      transferFrom(_from, _to, _tokenId);
    if (isContract(_to)){
      bytes4 returnValue = ERC721TokenReceiver(_to).onERC721Received(msg.sender, _from,
_tokenId, data);
      //위의 safeTransferFrom 와 같지만 bytes data 매개변수가 추가된다.
      // 프런트엔드쪽에서 만들 변수를 미리 지정해놓는다
      require(returnValue == 0x150b7a02);
    }
  }

  function approve(address _approved, uint256 _tokenId) public{
  // 이 함수는 권한을 받게되는 계정과 토큰아이디를 매개변수로 받는다
    address owner = ownerOf(_tokenId);
    // 토큰 소유자 계정을 불러온다
    require(_approved != owner);
    // 원한을 받게되는 계정이 소유자 계정이 아니여야 통과시킨다
    require(msg.sender == owner);
    //approve 를 호출한 계정(msg.sender)이 소유자 계정이어야 한다.
    tokenApprovals[_tokenId] = _approved;
  }

  function getApproved(uint256 _tokenId) public view returns (address){
    return tokenApprovals[_tokenId];
  }

  function setApprovalForAll(address _operator, bool _approved) public{
    //_operator는 모든 토큰을 대신 운용해줄 계정
    // _operator는 일반계정 contract 계정 둘다가능
    // 권한을 부여할지를 approve 를 이이동
    require (_operator != msg.sender);
    // _operator 계정이 현재 함수를 호출한 계정이 되어서는 안된다.
    operatorApprovals[msg.sender][_operator] = _approved;
    //함수를 호출한 계정이 오퍼레이터 계정에게 권한을 부여할 것인지 말것인지를
_approved 인자를 통해 mapping 에 저장
    // _approved 가 true 면 권한 부여, false 면 권한 취소.
  }

  function isApprovedForAll(address _owner, address _operator) public view returns (bool){
    // _owner와 _operator를 받고 bool을 리리턴하는
    // 토큰의 오너가 _operator에게 권한을 부여했는지 안했는지 확인하는 함수
    return operatorApprovals[_owner][_operator];
    // 토큰 소유자 계정을 키값으로 해서 _operator계정을 확인해서 ower가 _operator에게
권한을 줬는지 안줬는지 true or false를 리턴하게 한다.
  }

  function supportsInterface(bytes4 interfaceID) public view returns (bool) {
    return supportsInterfaces[interfaceID];
    //외부에서 확인하기 위해서 supportsInterface 함수에 interfaceID를 넣으면 true/false를 확
인할 수 있다.
  }

  function isContract(address _addr) private view returns (bool){
    //address가 contract계정이면 true 아니면 false로 리턴하는 함수
    uint256 size;
```

```
        assembly { size := extcodesize(_addr) }
        return size > 0;
        // extcodesize에 주소를 넣었을 때 0이면 일반계정이고 0보다 크면 컨트렉트계정

    }
}

contract Auction is ERC721TokenReceiver {
    //ERC721TokenReceiver 상속받음
    function onERC721Received(address _operator, address _from, uint256 _tokenId, bytes _data)
public returns(bytes4){
        return bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"));
    // safeTransferFrom 에서 if 조건과 require 조건을 contract Auction이 모두 충족하고 있다!!유
효성검사 통과가능!
    // token 이동 기능이 없다(테스트용이 | 기 때문)
    }
    function checkSupportsInterface(address _to, bytes4 interfaceID) public view returns (bool){
    //_to에는 ERC721Implementation contract의 주소값을 넘기고 interfaceID는 ERC721 interface
의 식별자를 넘길것이다.
        return ERC721Implementation(_to).supportsInterface(interfaceID);

    }
}
```

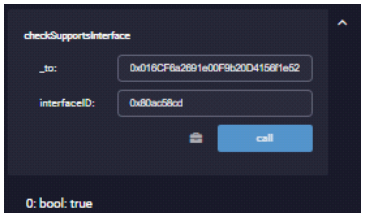


- auction 먼저 deploy하고

- ERC721도 deploy한다.
- contract가 2개가 되도록 한다



- ERC721implementation의 contract 주소를 auction의 chechSupportsInterface 함수의 -to에 입력한다.
- interfaceID는 0x80ac58cd(ERC721implementation 식별자)를 넣는다.



- ERC721implementation contract이 ERC721 interface를 구현하고 있다.
- 외부 contract(auction)에서 확인.