Chapter 12 ERC-721 & ERC-1155

ERC-721
소스코드 살펴보기

ERC-721 구현 소스코드 살펴보기

Openzeppelin 에서 구현한 ERC-721

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC721/ERC721.sol

```
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.7.0) (token/ERC721/ERC721.sol)

pragma solidity ^0.8.0;

import "./IERC721.sol";
import "./IERC721Receiver.sol";
import "./extensions/IERC721Metadata.sol";
import "../../utils/Address.sol";
import "../../utils/Context.sol";
import "../../utils/Strings.sol";
import "../../utils/introspection/ERC165.sol";

/**
 * @dev Implementation of https://eips.ethereum.org/EIPS/eip-721[ERC721] Non-Fungible Token Standard, including
 * the Metadata extension, but not including the Enumerable extension, which is available separately as
 * {ERC721Enumerable}.
 */
contract ERC721 is Context, ERC165, IERC721, IERC721Metadata {
    using Address for address;
    using Strings for uint256;

    // Token name
    string private _name;

    // Token symbol
    string private _symbol;

    // Mapping from token ID to owner address
    mapping(uint256 => address) private _owners;

    // Mapping owner address to token count
    mapping(address => uint256) private _balances;

    // Mapping from token ID to approved address
    mapping(uint256 => address) private _tokenApprovals;

    // Mapping from owner to operator approvals
    mapping(address => mapping(address => bool)) private _operatorApprovals;

    /**
     * @dev Initializes the contract by setting a `name` and a `symbol` to the token collection.
     */
    constructor(string memory name_, string memory symbol_) {
        _name = name_;
        _symbol = symbol_;
    }

```

ERC-721 구현
소스코드 살펴보기

_owners 변수에 각 Token ID 별 소유자가 저장됨

ex) Token ID 123 번의 소유자는
_owner[123] 에 저장되어있음

```
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.7.0) (token/ERC721/ERC721.sol)

pragma solidity ^0.8.0;

import "./IERC721.sol";
import "./IERC721Receiver.sol";
import "./extensions/IERC721Metadata.sol";
import "../../utils/Address.sol";
import "../../utils/Context.sol";
import "../../utils/Strings.sol";
import "../../utils/introspection/ERC165.sol";

/**
 * @dev Implementation of https://eips.ethereum.org/EIPS/eip-721[ERC721] Non-Fungible Token Standard, including
 * the Metadata extension, but not including the Enumerable extension, which is available separately as
 * {ERC721Enumerable}.
 */
contract ERC721 is Context, ERC165, IERC721, IERC721Metadata {
    using Address for address;
    using Strings for uint256;

    // Token name
    string private _name;

    // Token symbol
    string private _symbol;

    // Mapping from token ID to owner address
    mapping(uint256 => address) private _owners;

    // Mapping owner address to token count
    mapping(address => uint256) private _balances;

    // Mapping from token ID to approved address
    mapping(uint256 => address) private _tokenApprovals;

    // Mapping from owner to operator approvals
    mapping(address => mapping(address => bool)) private _operatorApprovals;

    /**
     * @dev Initializes the contract by setting a `name` and a `symbol` to the token collection.
     */
    constructor(string memory name_, string memory symbol_) {
        _name = name_;
        _symbol = symbol_;
    }

```

ERC-721 구현
소스코드 살펴보기

tokenURI 함수는 각 토큰에 대한 token URI 리턴

OpenZeppelin 에 구현된 코드의 경우 tokenURI는 **${baseURI}${tokenId}** 값을 리턴

ex1) baseURI 가 https://example.com/ 이고,
token ID 가 123이면,

tokenURI(123) 은
https://example.com/123 값을 리턴

```
/**
 * @dev See {IERC721Metadata-tokenURI}.
 */
function tokenURI(uint256 tokenId) public view virtual override returns (string memory) {
    _requireMinted(tokenId);

    string memory baseURI = _baseURI();
    return bytes(baseURI).length > 0 ? string(abi.encodePacked(baseURI, tokenId.toString())) : "";
}

/**
 * @dev Base URI for computing {tokenURI}. If set, the resulting URI for each
 * token will be the concatenation of the `baseURI` and the `tokenId`. Empty
 * by default, can be overridden in child contracts.
 */
function _baseURI() internal view virtual returns (string memory) {
    return "";
}
```

ownerOf

토큰의 소유자가 누구인지 리턴하는 함수

토큰이 존재하지 않는 경우 = 토큰 소유자가 Null Address (Address 가 0x0)

토큰이 존재하는 경우 = 토큰 소유자가 Null Address 가 아님 (Address 가 0x0 이 아님)

```
/**
 * @dev See {IERC721-ownerOf}.
 */
function ownerOf(uint256 tokenId) public view virtual override returns (address) {
    address owner = _owners[tokenId];
    require(owner != address(0), "ERC721: invalid token ID");
    return owner;
}
```

approve

내가 소유한 어떤 토큰을 특정 지갑이 사용할 수 있도록 허용

허용한다는 것은 소유권 변경 등 특정 지갑에게 특정 토큰을 이용할 수 있도록 허가해준다는 의미

구조 상 토큰 하나 당 하나의 지갑에만 허용 가능

```
/**
 * @dev See {IERC721-approve}.
 */
function approve(address to, uint256 tokenId) public virtual override {
    address owner = ERC721.ownerOf(tokenId);
    require(to != owner, "ERC721: approval to current owner");

    require(
        _msgSender() == owner || isApprovedForAll(owner, _msgSender()),
        "ERC721: approve caller is not token owner nor approved for all"
    );

    _approve(to, tokenId);
}
```

```
/**
 * @dev Approve `to` to operate on `tokenId`
 *
 * Emits an {Approval} event.
 */
function _approve(address to, uint256 tokenId) internal virtual {
    _tokenApprovals[tokenId] = to;
    emit Approval(ERC721.ownerOf(tokenId), to, tokenId);
}
```

setApprovalForAll

내가 가지고 있는 토큰 전체를 특정 지갑에게 허용

```
/**
 * @dev See {IERC721-setApprovalForAll}.
 */
function setApprovalForAll(address operator, bool approved) public virtual override {
    _setApprovalForAll(_msgSender(), operator, approved);
}
```

```
/**
 * @dev Approve `operator` to operate on all of `owner` tokens
 *
 * Emits an {ApprovalForAll} event.
 */
function _setApprovalForAll(
    address owner,
    address operator,
    bool approved
) internal virtual {
    require(owner != operator, "ERC721: approve to caller");
    _operatorApprovals[owner][operator] = approved;
    emit ApprovalForAll(owner, operator, approved);
}
```

setApprovalForAll

approve 는 내가 가지고 있는 어떤 하나의 토큰을 특정 계정에게 허용
(하나의 토큰에 대해서만 1회성으로 다른 주소에게 허용)

setApprovalForAll 는 내가 소유하고 있는 토큰을 특정 계정에게 허용
(지갑이 소유한 모든 토큰에 대해서 허용)

```
/**
 * @dev See {IERC721-setApprovalForAll}.
 */
function setApprovalForAll(address operator, bool approved) public virtual override {
    _setApprovalForAll(_msgSender(), operator, approved);
}
```

```
/**
 * @dev Approve `operator` to operate on all of `owner` tokens
 *
 * Emits an {ApprovalForAll} event.
 */
function _setApprovalForAll(
    address owner,
    address operator,
    bool approved
) internal virtual {
    require(owner != operator, "ERC721: approve to caller");
    _operatorApprovals[owner][operator] = approved;
    emit ApprovalForAll(owner, operator, approved);
}
```

setApprovalForAll

왜 다른 지갑에 허용하는 것이 필요하는가?
여러가지 목적이 있을 수 있겠지만 예를 들어 NFT 거래를 컨트랙트를 통해 한다고 했을 때 자신이 소유한 NFT 를 컨트랙트에게 허용을 해주어야지만 거래 가능.

OpenSea 등 NFT 마켓에서 사용자들 간 P2P 거래 등을 지원하기 위해서는 승인 필요

```
/**
 * @dev See {IERC721-setApprovalForAll}.
 */
function setApprovalForAll(address operator, bool approved) public virtual override {
    _setApprovalForAll(_msgSender(), operator, approved);
}
```

```
/**
 * @dev Approve `operator` to operate on all of `owner` tokens
 *
 * Emits an {ApprovalForAll} event.
 */
function _setApprovalForAll(
    address owner,
    address operator,
    bool approved
) internal virtual {
    require(owner != operator, "ERC721: approve to caller");
    _operatorApprovals[owner][operator] = approved;
    emit ApprovalForAll(owner, operator, approved);
}
```

transferFrom

NFT 전송을 위한 함수
(NFT 소유권 이전)

“from 에서 to 로 tokenId 의 소유권을 넘긴다”

```
/**
 * @dev See {IERC721-transferFrom}.
 */
function transferFrom(
    address from,
    address to,
    uint256 tokenId
) public virtual override {
    //solhint-disable-next-line max-line-length
    require(_isApprovedOrOwner(_msgSender(), tokenId), "ERC721: caller is not token owner nor approved");

    _transfer(from, to, tokenId);
}
```

transferFrom

함수를 호출한 사람이

토큰을 소유한 **소유권자(Owner)** 또는
그 토큰의 사용을 **허가받은(Approved)** 자인
경우에만

토큰 소유권 이전 가능

```
/**
 * @dev See {IERC721-transferFrom}.
 */
function transferFrom(
    address from,
    address to,
    uint256 tokenId
) public virtual override {
    //solhint-disable-next-line max-line-length
    require(_isApprovedOrOwner(_msgSender(), tokenId), "ERC721: caller is not token owner nor approved");

    _transfer(from, to, tokenId);
}
```

transferFrom

토큰의 소유권이 from → to 로 이전되므로
from 주소는 토큰의 소유자이어야 함

```
function _transfer(
    address from,
    address to,
    uint256 tokenId
) internal virtual {
    require(ERC721.ownerOf(tokenId) == from, "ERC721: transfer from incorrect owner");
    require(to != address(0), "ERC721: transfer to the zero address");

    _beforeTokenTransfer(from, to, tokenId);

    // Clear approvals from the previous owner
    delete _tokenApprovals[tokenId];

    _balances[from] -= 1;
    _balances[to] += 1;
    _owners[tokenId] = to;

    emit Transfer(from, to, tokenId);

    _afterTokenTransfer(from, to, tokenId);
}
```

transferFrom

Null Address (주소 0x00000..000) 으로 이전 불가

Why?
발행되지 않은 토큰의 경우 소유자가 없으므로 0 값이 저장
반대로 소유자가 Null Address 면 토큰이 없다는 것을 의미

Null Address 로 보낸다는 것은 토큰이 존재했는데 없어졌다는 것을 의미하므로 전송 금지

```
function _transfer(
    address from,
    address to,
    uint256 tokenId
) internal virtual {
    require(ERC721.ownerOf(tokenId) == from, "ERC721: transfer from incorrect owner");
    require(to != address(0), "ERC721: transfer to the zero address");

    _beforeTokenTransfer(from, to, tokenId);

    // Clear approvals from the previous owner
    delete _tokenApprovals[tokenId];

    _balances[from] -= 1;
    _balances[to] += 1;
    _owners[tokenId] = to;

    emit Transfer(from, to, tokenId);

    _afterTokenTransfer(from, to, tokenId);
}
```

transferFrom

토큰 전송 전 어떤 기능 수행.

사용자가 _beforeTokenTransfer 함수를 상속받아서 구현하면 그 함수가 토큰 전송 전 실행됨.

기본적으로는 아무 기능도 수행하지 않음.

```
function _transfer(
    address from,
    address to,
    uint256 tokenId
) internal virtual {
    require(ERC721.ownerOf(tokenId) == from, "ERC721: transfer from incorrect owner");
    require(to != address(0), "ERC721: transfer to the zero address");

    _beforeTokenTransfer(from, to, tokenId);

    // Clear approvals from the previous owner
    delete _tokenApprovals[tokenId];

    _balances[from] -= 1;
    _balances[to] += 1;
    _owners[tokenId] = to;

    emit Transfer(from, to, tokenId);

    _afterTokenTransfer(from, to, tokenId);
}
```

transferFrom

NFT 주인은 자신의 토큰을 다른 주소가
사용할 수 있도록 허용(Approval) 가능

주인이 이전되면 이 토큰의 허가를 초기화시킴.

(원래 주인은 특정 지갑에게 NFT 를
허용해줬지만 새 주인은 NFT 를 그 지갑에게
허용해주고 싶지 않을수도 있으므로)

```
function _transfer(
    address from,
    address to,
    uint256 tokenId
) internal virtual {
    require(ERC721.ownerOf(tokenId) == from, "ERC721: transfer from incorrect owner");
    require(to != address(0), "ERC721: transfer to the zero address");

    _beforeTokenTransfer(from, to, tokenId);

    // Clear approvals from the previous owner
    delete _tokenApprovals[tokenId];

    _balances[from] -= 1;
    _balances[to] += 1;
    _owners[tokenId] = to;

    emit Transfer(from, to, tokenId);

    _afterTokenTransfer(from, to, tokenId);
}
```

transferFrom

_balances 에는 컨트랙트 내에서 특정 주소가 보유하고 있는 NFT 수량을 저장함

따라서 소유권이 이전되었으면 기존 소유권자의 _balance 값은 1 을 빼고, 새로운 소유권자의 _balance 값은 1 추가

그 후
_owner 값을 변경하여 소유권 변경

```
function _transfer(
    address from,
    address to,
    uint256 tokenId
) internal virtual {
    require(ERC721.ownerOf(tokenId) == from, "ERC721: transfer from incorrect owner");
    require(to != address(0), "ERC721: transfer to the zero address");

    _beforeTokenTransfer(from, to, tokenId);

    // Clear approvals from the previous owner
    delete _tokenApprovals[tokenId];

    _balances[from] -= 1;
    _balances[to] += 1;
    _owners[tokenId] = to;

    emit Transfer(from, to, tokenId);

    _afterTokenTransfer(from, to, tokenId);
}
```

transferFrom

Transfer 이벤트 발생

이벤트(로깅)는 전송 등 추적 용도로 사용

```
function _transfer(
    address from,
    address to,
    uint256 tokenId
) internal virtual {
    require(ERC721.ownerOf(tokenId) == from, "ERC721: transfer from incorrect owner");
    require(to != address(0), "ERC721: transfer to the zero address");

    _beforeTokenTransfer(from, to, tokenId);

    // Clear approvals from the previous owner
    delete _tokenApprovals[tokenId];

    _balances[from] -= 1;
    _balances[to] += 1;
    _owners[tokenId] = to;

    emit Transfer(from, to, tokenId);

    _afterTokenTransfer(from, to, tokenId);
}
```

transferFrom

토큰 전송 후 어떤 기능 수행.

_beforeTokenTransfer 함수와 비슷하게 개발자가 함수 상속받아서 토큰 전송 후 특정 코드를 실행할 수 있는 확장용 함수.

함수 상속받지 않으면 아무 기능도 수행하지 않음.

```
function _transfer(
    address from,
    address to,
    uint256 tokenId
) internal virtual {
    require(ERC721.ownerOf(tokenId) == from, "ERC721: transfer from incorrect owner");
    require(to != address(0), "ERC721: transfer to the zero address");

    _beforeTokenTransfer(from, to, tokenId);

    // Clear approvals from the previous owner
    delete _tokenApprovals[tokenId];

    _balances[from] -= 1;
    _balances[to] += 1;
    _owners[tokenId] = to;

    emit Transfer(from, to, tokenId);

    _afterTokenTransfer(from, to, tokenId);
}
```

safeTransferFrom

safeTransferFrom 의 경우 transferFrom 과 동일하나

토큰 전송 후 전송된 주소가 컨트랙트 주소일 경우

컨트랙트에 콜백 함수 호출하여 확인
컨트랙트에서 콜백 함수에 대해서 적절하게 처리해주어야지만 전송 성공

```
/**
 * @dev See {IERC721-safeTransferFrom}.
 */
function safeTransferFrom(
    address from,
    address to,
    uint256 tokenId,
    bytes memory data
) public virtual override {
    require(_isApprovedOrOwner(_msgSender(), tokenId), "ERC721: caller is not token owner nor approved");
    _safeTransfer(from, to, tokenId, data);
}

function _safeTransfer(
    address from,
    address to,
    uint256 tokenId,
    bytes memory data
) internal virtual {
    _transfer(from, to, tokenId);
    require(_checkOnERC721Received(from, to, tokenId, data), "ERC721: transfer to non ERC721Receiver implementer");
}
```

safeTransferFrom

NFT 가 컨트랙트로 전송될 경우 컨트랙트에 영원히 묶여버리는 경우가 생김. (컨트랙트에 출금 기능이 없을 경우)

이런 문제를 해결하기 위해 추가된 함수.

```
/**
 * @dev See {IERC721-safeTransferFrom}.
 */
function safeTransferFrom(
    address from,
    address to,
    uint256 tokenId,
    bytes memory data
) public virtual override {
    require(_isApprovedOrOwner(_msgSender(), tokenId), "ERC721: caller is not token owner nor approved");
    _safeTransfer(from, to, tokenId, data);
}

function _safeTransfer(
    address from,
    address to,
    uint256 tokenId,
    bytes memory data
) internal virtual {
    _transfer(from, to, tokenId);
    require(_checkOnERC721Received(from, to, tokenId, data), "ERC721: transfer to non ERC721Receiver implementer");
}
```