Homework 9 - ERC721 Answer

See this gist

```
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts for Cairo v0.2.1
(token/erc721/ERC721_Mintable_Burnable.cairo)

%lang starknet

from starkware.cairo.common.cairo_builtins import HashBuiltin
from starkware.cairo.common.uint256 import Uint256, uint256_add,
uint256_signed_le
from openzeppelin.access.ownable import Ownable
from openzeppelin.introspection.ERC165 import ERC165
from openzeppelin.token.erc721.library import ERC721
from starkware.starknet.common.syscalls import get_caller_address
from starkware.cairo.common.math import assert_le

//
// Constructor
//

@storage_var
func counter() -> (idx: Uint256) {
}

@storage_var
func og_owner(tokenId: Uint256) -> (o: felt) {
}

@constructor
func constructor{syscall_ptr: felt*, pedersen_ptr: HashBuiltin*,
range_check_ptr}(
    name: felt, symbol: felt, owner: felt
) {
    ERC721.initializer(name, symbol);
    Ownable.initializer(owner);
    counter.write(Uint256(0, 0));
    return ();
```

```
}

//
// Getters
//

@view
func getOriginalOwner{syscall_ptr: felt*, pedersen_ptr: HashBuiltin*,
range_check_ptr}(
    tokenId: Uint256
) -> (address: felt) {
    let (owner) = og_owner.read(tokenId);
    return (owner,);
}

@view
func getCounter{syscall_ptr: felt*, pedersen_ptr: HashBuiltin*,
range_check_ptr}() -> (
    idx: Uint256
) {
    let (current_counter) = counter.read();
    return (current_counter,);
}

@view
func supportsInterface{syscall_ptr: felt*, pedersen_ptr: HashBuiltin*,
range_check_ptr}(
    interfaceId: felt
) -> (success: felt) {
    let (success) = ERC165.supports_interface(interfaceId);
    return (success,);
}

@view
func name{syscall_ptr: felt*, pedersen_ptr: HashBuiltin*, range_check_ptr}
() -> (name: felt) {
    let (name) = ERC721.name();
    return (name,);
}

@view
func symbol{syscall_ptr: felt*, pedersen_ptr: HashBuiltin*,
range_check_ptr}() -> (symbol: felt) {
    let (symbol) = ERC721.symbol();
    return (symbol,);
}

@view
func balanceOf{syscall_ptr: felt*, pedersen_ptr: HashBuiltin*,
range_check_ptr}(owner: felt) -> (
    balance: Uint256
) {
    let (balance: Uint256) = ERC721.balance_of(owner);
    return (balance,);
}

@view
func ownerOf{syscall_ptr: felt*, pedersen_ptr: HashBuiltin*,
range_check_ptr}(tokenId: Uint256) -> (
    owner: felt
) {
    let (owner: felt) = ERC721.owner_of(tokenId);
    return (owner,);
}

@view
func getApproved{syscall_ptr: felt*, pedersen_ptr: HashBuiltin*,
range_check_ptr}(
    tokenId: Uint256
) -> (approved: felt) {
    let (approved: felt) = ERC721.get_approved(tokenId);
    return (approved,);
}

@view
func isApprovedForAll{syscall_ptr: felt*, pedersen_ptr: HashBuiltin*,
range_check_ptr}(
    owner: felt, operator: felt
) -> (isApproved: felt) {
    let (isApproved: felt) = ERC721.is_approved_for_all(owner, operator);
    return (isApproved,);
}

@view
func tokenURI{syscall_ptr: felt*, pedersen_ptr: HashBuiltin*,
range_check_ptr}(
    tokenId: Uint256
) -> (tokenURI: felt) {
    let (tokenURI: felt) = ERC721.token_uri(tokenId);
    return (tokenURI,);
}

@view
func owner{syscall_ptr: felt*, pedersen_ptr: HashBuiltin*,
range_check_ptr}() -> (owner: felt) {
    let (owner: felt) = Ownable.owner();
    return (owner,);
}

//
// Externals
//

@external
func approve{pedersen_ptr: HashBuiltin*, syscall_ptr: felt*,
range_check_ptr}(
    to: felt, tokenId: Uint256
) {
    ERC721.approve(to, tokenId);
    return ();
}

@external
func setApprovalForAll{syscall_ptr: felt*, pedersen_ptr: HashBuiltin*,
range_check_ptr}(
    operator: felt, approved: felt
) {
    ERC721.set_approval_for_all(operator, approved);
    return ();
}

@external
func transferFrom{pedersen_ptr: HashBuiltin*, syscall_ptr: felt*,
range_check_ptr}(
    from_: felt, to: felt, tokenId: Uint256
) {
    ERC721.transfer_from(from_, to, tokenId);
    return ();
}
```

```
@external
func safeTransferFrom{pedersen_ptr: HashBuiltin*, syscall_ptr: felt*,
range_check_ptr}(
    from_: felt, to: felt, tokenId: Uint256, data_len: felt, data: felt*
) {
    ERC721.safe_transfer_from(from_, to, tokenId, data_len, data);
    return ();
}

// Solution
//########################################################################
#############

@external
func mint{pedersen_ptr: HashBuiltin*, syscall_ptr: felt*, range_check_ptr}
(to: felt) {
    Ownable.assert_only_owner();

    // Read counter
    let (tokenId) = counter.read();

    // Add 1
    let (new_token_id, _) = uint256_add(tokenId, Uint256(1, 0));

    // increment
    counter.write(new_token_id);

    // Add original hash TODO diff register
    og_owner.write(new_token_id, to);

    // Mint
    ERC721._mint(to, new_token_id);
    return ();
}

@external
func burn{pedersen_ptr: HashBuiltin*, syscall_ptr: felt*, range_check_ptr}
(tokenId: Uint256) {
    ERC721.assert_only_token_owner(tokenId);
    ERC721._burn(tokenId);
    return ();
}
```

```
@external
func setTokenURI{pedersen_ptr: HashBuiltin*, syscall_ptr: felt*,
range_check_ptr}(
    tokenId: Uint256, tokenURI: felt
) {
    Ownable.assert_only_owner();
    ERC721._set_token_uri(tokenId, tokenURI);
    return ();
}

@external
func transferOwnership{syscall_ptr: felt*, pedersen_ptr: HashBuiltin*,
range_check_ptr}(
    newOwner: felt
) {
    Ownable.transfer_ownership(newOwner);
    return ();
}

@external
func renounceOwnership{syscall_ptr: felt*, pedersen_ptr: HashBuiltin*,
range_check_ptr}() {
    Ownable.renounce_ownership();
    return ();
}
```