# 1.https://stackoverflow.com/questions/67317392/how-to-transfer-a-nft-from-one-account-to-another-using-erc721

**T:**How to transfer a NFT from one account to another using ERC721?

**Q:**I'm writing an NFT smart contract using the OpenZeppelin ERC721Full contract. I'm able to mint NFTs, but I want to have a button that enables them to be bought. I'm trying writing this function:  
  
function buyNFT(uint \_id) public payable{ //Get NFT owner address address payable \_seller = ownerOf(\_id); // aprove nft sell approve(\_seller, \_id); setApprovalForAll(msg.sender, true); //transfer NFT transferFrom(\_seller, msg.sender, \_id); // transfer price in ETH address(\_seller).transfer(msg.value); emit NftBought(\_seller, msg.sender, msg.value); }  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
This does not work because function approve must be called by the owner or an already approved address. I have no clue on how a buy function should be built. I know that I must use some requirements but first I want the function to work on tests and then I'll write the requirements.  
  
How should a buy function be coded? Because the only solution I have found is to overwrite the approve function and omit the require of who can call this function. But it looks like it isn't the way it should be done.  
  
Thank you!

3 **Answer**

**A1:**You can use just the \_transfer() function, see my buy() function for an example of implementation.  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
The approvals for sale can be done using a custom mapping - in my example tokenIdToPrice. If the value is non-zero, the token ID (mapping key) is for sale.  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
This is a basic code that allows selling an NTF. Feel free to expand on my code to allow "give away for free", "whitelist buyers" or any other feature.  
  
pragma solidity ^0.8.4;import 'https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC721/ERC721.sol';contract MyToken is ERC721 { event NftBought(address \_seller, address \_buyer, uint256 \_price); mapping (uint256 => uint256) public tokenIdToPrice; constructor() ERC721('MyToken', 'MyT') { \_mint(msg.sender, 1); } function allowBuy(uint256 \_tokenId, uint256 \_price) external { require(msg.sender == ownerOf(\_tokenId), 'Not owner of this token'); require(\_price > 0, 'Price zero'); tokenIdToPrice[\_tokenId] = \_price; } function disallowBuy(uint256 \_tokenId) external { require(msg.sender == ownerOf(\_tokenId), 'Not owner of this token'); tokenIdToPrice[\_tokenId] = 0; } function buy(uint256 \_tokenId) external payable { uint256 price = tokenIdToPrice[\_tokenId]; require(price > 0, 'This token is not for sale'); require(msg.value == price, 'Incorrect value'); address seller = ownerOf(\_tokenId); \_transfer(seller, msg.sender, \_tokenId); tokenIdToPrice[\_tokenId] = 0; // not for sale anymore payable(seller).transfer(msg.value); // send the ETH to the seller emit NftBought(seller, msg.sender, msg.value); }}  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
How to simulate the sale:  
  
 ● The contract deployer (msg.sender) gets token ID 1.  
  
 ● Execute allowBuy(1, 2) that will allow anyone to buy token ID 1 for 2 wei.  
  
 ● From a second address, execute buy(1) sending along 2 wei, to buy the token ID 1.  
  
 ● Call (the parent ERC721) function ownerOf(1) to validate that the owner is now the second address.

**C1:**I've used \_transferFrom(seller, msg.sender, \_tokenId); insted of \_transfer(seller, msg.sender, \_tokenId); because I'm using ERC721Full but it worked nicely. Thank you!

**C2:**What is the line that starts with "mapping" doing?

**C3:**@ianwt A mapping is a dictionary-like datatype. You can easily retrieve a value by its key, but you cannot retrieve a key by a value. Note that the keys have to be unique, the values don't... In the example above, the key is the token ID, and the value is the token price. So in this case its easy to query a token price by its ID.

**C4:**Got it, thank you! what is the value of this mapping when the contract is initially used to mint an NFT? Is it 0? That is, does allowBuy have to be explicitly called with a positive value before the NFT is purchasable? Thanks again

**C5:**@ianwt Exactly. Default value for each key is 0. And because the buy() function requires price > 0 (i.e. non-default value), you effectively need to invoke allowBuy() before the NFT is purchasable.

**A2:**If you let anyone call the approve function, it would allow anyone to approve themselves to take NFTs! The purpose of approve is to give the owner of an asset the ability to give someone else permission to transfer that asset as if it was theirs.  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
The basic premise of any sale is that you want to make sure that you get paid, and that the buyer receives the goods in return for the sale. Petr Hedja's solution takes care of this by having the buy function not only transfer the NFT, but also include the logic for sending the price of the token. I'd like to recommend a similar structure with a few changes. One is so that the function will also work with ERC20 tokens, the other is to prevent an edge case where if gas runs out during execution, the buyer could end up with their NFT for free. This is building on his answer, though, and freely uses some of the code in that answer for architecture.  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
Ether can still be set as the accepted currency by inputting the zero address (address(0)) as the contract address of the token.  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
If the sale is in an ERC20 token, the buyer will need to approve the NFT contract to spend the amount of the sale since the contract will be pulling the funds from the buyer's account directly.  
  
pragma solidity ^0.8.4;import 'https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC721/ERC721.sol';import 'https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/IERC20.sol';contract MyToken is ERC721 { event NftBought(address \_seller, address \_buyer, uint256 \_price); mapping (uint256 => uint256) public tokenIdToPrice; mapping (uint256 => address) public tokenIdToTokenAddress; constructor() ERC721('MyToken', 'MyT') { \_mint(msg.sender, 1); } function setPrice(uint256 \_tokenId, uint256 \_price, address \_tokenAddress) external { require(msg.sender == ownerOf(\_tokenId), 'Not owner of this token'); tokenIdToPrice[\_tokenId] = \_price; tokenIdToTokenAddress[\_tokenId] = \_tokenAddress; } function allowBuy(uint256 \_tokenId, uint256 \_price) external { require(msg.sender == ownerOf(\_tokenId), 'Not owner of this token'); require(\_price > 0, 'Price zero'); tokenIdToPrice[\_tokenId] = \_price; } function disallowBuy(uint256 \_tokenId) external { require(msg.sender == ownerOf(\_tokenId), 'Not owner of this token'); tokenIdToPrice[\_tokenId] = 0; } function buy(uint256 \_tokenId) external payable { uint256 price = tokenIdToPrice[\_tokenId]; require(price > 0, 'This token is not for sale'); require(msg.value == price, 'Incorrect value'); address seller = ownerOf(\_tokenId); address tokenAddress = tokenIdToTokenAddress[\_tokenId]; if(address != address(0){ IERC20 tokenContract = IERC20(tokenAddress); require(tokenContract.transferFrom(msg.sender, address(this), price), "buy: payment failed"); } else { payable(seller).transfer(msg.value); } \_transfer(seller, msg.sender, \_tokenId); tokenIdToPrice[\_tokenId] = 0; emit NftBought(seller, msg.sender, msg.value); }}  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]

**C1:**There's something wrong with if(address != address(0){, it's missing parenthesis and the comparison doesn't seem right.

**C2:**The Renaissance, can you explain to me the tokenIdToTokenAddress and \_tokenAddress and why you created the setPrice function? I suppose it's to keep track of what ERC20 currency the NFT is for sale.

**C3:**Hey I have made a question elaborating on your answer here, I wondered if @The Renaissance might be able to have a look and answer it? Thanks stackoverflow.com/questions/69193720/…

**A3:**// mapping is for fast lookup. the longer operation, the more gasmapping(uint => NftItem) private \_idToNftItem;function buyNft(uint tokenId) public payable{ uint price=\_idToNftItem[tokenId].price; // this is set in erc721 contract // Since contracts are inheriting, I want to make sure I use this method in ERC721 address owner=ERC721.ownerOf(tokenId); require(msg.sender!=owner,"You already own this nft"); require(msg.value==price,"Please submit the asking price"); // since this is purchased, it is not for sale anymore \_idToNftItem[tokenId].isListed=false; \_listedItems.decrement(); // this is defined in ERC721 // this already sets owner \_owners[tokenId] = msg.sender; \_transfer(owner,msg.sender,tokenId); payable(owner).transfer(msg.value); }  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
this is Nft struct  
  
struct NftItem{ uint tokenId; uint price; // creator and owner are not same. creator someone who minted. creator does not change address creator; bool isListed; }  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]