# 7.https://stackoverflow.com/questions/71063058/my-buy-function-in-smart-contract-file-keep-making-error

**T:**My "Buy" function in smart contract file keep making error

**Q:**This is my solidity file for NFT marketplace.  
  
// SPDX-License-Identifier: MITpragma solidity ^0.8.0;import "@openzeppelin/contracts/token/ERC721/ERC721.sol";import "@openzeppelin/contracts/access/Ownable.sol";import "@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol";import "@openzeppelin/contracts/utils/Counters.sol"; contract NFT is ERC721URIStorage,Ownable { using Counters for Counters.Counter; Counters.Counter private \_tokenIds; address payable public \_owner; mapping(address => uint[]) public addressToTokenArray; mapping(uint256 => bool) public forSale; mapping(uint256 => uint256) public tokenIdToPrice; event Minting(address \_owner, uint256 \_tokenId, uint256 \_price); event Purchase(address \_seller, address \_buyer, uint256 \_price); event Remove(uint256 \_tokenId, uint[] beforeBuy, uint[] afterBuy); constructor() ERC721("TeddyBear", "TEDDY") { } function mint(string memory \_tokenURI, uint256 \_price) public onlyOwner returns (bool) { \_tokenIds.increment(); uint256 newItemId = \_tokenIds.current(); tokenIdToPrice[newItemId] = \_price; if(addressToTokenArray[msg.sender].length !=1){ addressToTokenArray[msg.sender].push(newItemId); }else{ addressToTokenArray[msg.sender] = [newItemId]; } \_mint(msg.sender, newItemId); \_setTokenURI(newItemId, \_tokenURI); emit Minting(msg.sender, newItemId, \_price); return true; } // 토큰의 주인이 판매 하는 함수 function sell(uint256 \_tokenId, uint256 \_price) external { require(msg.sender == ownerOf(\_tokenId), 'Not owner of this token'); require(\_price > 0, 'Price zero'); tokenIdToPrice[\_tokenId] = \_price; forSale[\_tokenId] = true; } // 토큰의 주인이 판매를 취하하는 함수 function stopSell(uint256 \_tokenId) external { require(msg.sender == ownerOf(\_tokenId), 'Not owner of this token'); forSale[\_tokenId] = false; } // function remove(uint[] memory array, uint index) public pure returns(uint[] memory) { // if (index >= array.length) return array; // for (uint i = index; i<array.length-1; i++){ // array[i] = array[i+1]; // } // delete array[array.length-1]; // return array; // } function buy(uint256 \_tokenId, uint256 sendAmount) external payable { uint256 price = tokenIdToPrice[\_tokenId]; bool isOnSale = forSale[\_tokenId]; require(isOnSale, 'This token is not for sale'); require(sendAmount == price, 'Incorrect value'); address seller = ownerOf(\_tokenId); require(seller == ownerOf(\_tokenId), 'Seller and Owner is not same'); // uint[] memory beforeBuy = addressToTokenArray[seller]; // // for(uint i=0;i<addressToTokenArray[seller].length;i++){ // // if(\_tokenId == addressToTokenArray[seller][i]){ // // remove(addressToTokenArray[seller],i); // // } // // } // uint[] memory afterBuy = addressToTokenArray[seller]; // emit Remove(\_tokenId, beforeBuy, afterBuy); addressToTokenArray[msg.sender] = [\_tokenId]; safeTransferFrom(seller, msg.sender, \_tokenId); forSale[\_tokenId] = true; payable(seller).transfer(sendAmount); // send the ETH to the seller emit Purchase(seller, msg.sender, sendAmount); } function getPrice(uint256 \_tokenId) public view returns (uint256){ uint256 price = tokenIdToPrice[\_tokenId]; return price; } function isSale(uint256 \_tokenId) public view returns (bool){ bool isOnSale = forSale[\_tokenId]; return isOnSale; } function getMyTokenId() public view returns (uint[] memory){ uint[] memory myTokens = addressToTokenArray[msg.sender]; return myTokens; }}  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
Among functions up there, the buy function does not emit an error when I compile the .sol file, but after I deploy this contract and send transaction for "buy" function it keeps making this error.  
  
I just want to know where I should fix it and if there is any better idea for other functions, feel free to let me know... many thanks

1 **Answer**

**A1:**Most likely it is failing here:  
  
 safeTransferFrom(seller, msg.sender, \_tokenId);  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
If you check the ERC721 contract, safeTransferFrom eventually calls this:  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
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); // \*\*\*\*\*\* HERE IS THE ISSUE \*\*\*\*\* \_approve(address(0), tokenId); \_balances[from] -= 1; \_balances[to] += 1; \_owners[tokenId] = to; emit Transfer(from, to, tokenId); \_afterTokenTransfer(from, to, tokenId); }  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
If your contract is going to transfer a token on behalf of owner, owner has to approve first.  
  
so from the seller's contract, this should be called:  
  
function \_approve(address to, uint256 tokenId) internal virtual { \_tokenApprovals[tokenId] = to; emit Approval(ERC721.ownerOf(tokenId), to, tokenId); }  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
tokenApprovals is a mapping that keeps track of which tokens can be transferred.  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
in order to test which function call is causing error, place this requirement statement require(sendAmount == price, 'Incorrect value'); right before the function. and pass an incorrect value and you will get an error : 'Incorrect value'  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
Then put that require statement after the function, and pass a wrong value, if this require does send you error, you can be sure that function is causing the error

**C1:**Thanks Yilmaz, I nearly get lost but finally get a hint for this problem. I have to try this right away.