# 324.https://stackoverflow.com/questions/71597113/erc1155-sell-buy-nft-solidity

**T:**ERC1155 Sell/Buy NFT Solidity

**Q:**My contract for ERC1155 marketplace to mint buy and sell the NFT.  
  
The nft is getting minted , However the NFT is not showing in market place and not able to purchase. I am facing this error.I have also applied setApprovedforAll method while minting still no help.  
  
 Should create and execute market sales: Error: VM Exception while processing transaction: reverted with reason string 'ERC1155: caller is not owner nor approved' at NFT1155.balanceOf (@openzeppelin/contracts/token/ERC1155/ERC1155.sol:71) at NFT1155.isApprovedForAll (@openzeppelin/contracts/token/ERC1155/ERC1155.sol:110) at NFT1155.createMarketSale (contracts/NFT1155.sol:165) at async HardhatNode.\_mineBlockWithPendingTxs (node\_modules/hardhat/src/internal/hardhat-network/provider/node.ts:1772:23) at async HardhatNode.mineBlock (node\_modules/hardhat/src/internal/hardhat-network/provider/node.ts:466:16) at async EthModule.\_sendTransactionAndReturnHash (node\_modules/hardhat/src/internal/hardhat-network/provider/modules/eth.ts:1496:18) at async HardhatNetworkProvider.request (node\_modules/hardhat/src/internal/hardhat-network/provider/provider.ts:118:18) at async EthersProviderWrapper.send (node\_modules/@nomiclabs/hardhat-ethers/src/internal/ethers-provider-wrapper.ts:13:20)  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
My contract for ERC1155 marketplace to mint buy and sell the NFT.  
  
// SPDX-License-Identifier: MITpragma solidity ^0.8.11;import "@openzeppelin/contracts/token/ERC1155/ERC1155.sol";import "@openzeppelin/contracts/access/Ownable.sol";import "@openzeppelin/contracts/token/ERC1155/extensions/ERC1155Supply.sol";import "@openzeppelin/contracts/utils/Counters.sol";import "hardhat/console.sol";contract NFT1155 is ERC1155, Ownable, ERC1155Supply { //contract address goes here and id will be dynamic and will be passed in the \_mint function calls //example https://ipfs.io/ipfs/QmT51bbxTbSiYGcF2X39sG6DGYyAX2413A1sZfiACMgJGP?filename={id}.json //if the if id 1 then https://ipfs.io/ipfs/QmT51bbxTbSiYGcF2X39sG6DGYyAX2413A1sZfiACMgJGP?filename=1.json will return the data that needs to be minted constructor() ERC1155("") {} mapping(uint256 => string) internal \_tokenURIs; mapping(uint256 => MarketItem) private idToMarketItem; Counters.Counter private \_itemsSold; struct MarketItem { uint256 tokenId; address payable seller; address payable owner; uint256 price; bool sold; } event MarketItemCreated( uint256 indexed tokenId, address seller, address owner, uint256 price, bool sold ); using Counters for Counters.Counter; Counters.Counter private \_tokenIds; //To chnage the URL String after the contract is deployed function setURI(string memory newuri) public onlyOwner { \_setURI(newuri); } function mintToken( string memory tokenURI, uint256 amount, uint256 price ) public returns (uint256) { uint256 newItemId = \_tokenIds.current(); \_mint(address(this), newItemId, amount, ""); \_setTokenUri(newItemId, tokenURI); //createMarketItem(newItemId, price, amount); \_tokenIds.increment(); return newItemId; } function createMarketItem( uint256 tokenId, uint256 price, uint256 amount ) private { require(price > 0, "Price must be at least 1 wei"); idToMarketItem[tokenId] = MarketItem( tokenId, payable(msg.sender), payable(address(this)), price, false ); setApprovalForAll(address(this), true); safeTransferFrom(msg.sender, address(this), tokenId, amount, ""); emit MarketItemCreated( tokenId, msg.sender, address(this), price, false ); } function onERC1155Received( address \_operator, address \_from, uint256 \_id, uint256 \_value, bytes calldata \_data ) external returns (bytes4) { return bytes4( keccak256( "onERC1155Received(address,address,uint256,uint256,bytes)" ) ); } function \_setTokenUri(uint256 tokenId, string memory tokenURI) private { \_tokenURIs[tokenId] = tokenURI; } function mintBatch( address to, uint256[] memory ids, uint256[] memory amounts, bytes memory data ) public onlyOwner { \_mintBatch(to, ids, amounts, data); } // The following functions are overrides required by Solidity. function \_beforeTokenTransfer( address operator, address from, address to, uint256[] memory ids, uint256[] memory amounts, bytes memory data ) internal override(ERC1155, ERC1155Supply) { super.\_beforeTokenTransfer(operator, from, to, ids, amounts, data); } /\* allows someone to resell a token they have purchased \*/ function resellToken( uint256 tokenId, uint256 price, uint256 amount ) public payable { require( idToMarketItem[tokenId].owner == msg.sender, "Only item owner can perform this operation" ); idToMarketItem[tokenId].sold = false; idToMarketItem[tokenId].price = price; idToMarketItem[tokenId].seller = payable(msg.sender); idToMarketItem[tokenId].owner = payable(address(this)); \_itemsSold.decrement(); safeTransferFrom(msg.sender, address(this), tokenId, amount, ""); } /\* Creates the sale of a marketplace item \*/ /\* Transfers ownership of the item, as well as funds between parties \*/ function createMarketSale(uint256 tokenId, uint256 amount) public payable { uint256 price = idToMarketItem[tokenId].price; address seller = idToMarketItem[tokenId].seller; console.log( " ~ file: NFT1155.sol ~ line 147 ~ createMarketSale ~ price", msg.value, price ); // require( // msg.value == price, // "Please submit the asking price in order to complete the purchase" // ); idToMarketItem[tokenId].owner = payable(msg.sender); idToMarketItem[tokenId].sold = true; idToMarketItem[tokenId].seller = payable(address(0)); \_itemsSold.increment(); safeTransferFrom(address(this), msg.sender, tokenId, amount, ""); setApprovalForAll(address(this), true); // payable(owner).transfer(listingPrice); payable(seller).transfer(msg.value); } /\* Returns all unsold market items \*/ function fetchMarketItems() public view returns (MarketItem[] memory) { uint256 itemCount = \_tokenIds.current(); uint256 unsoldItemCount = \_tokenIds.current() - \_itemsSold.current(); uint256 currentIndex = 0; MarketItem[] memory items = new MarketItem[](unsoldItemCount); for (uint256 i = 0; i < itemCount; i++) { if (idToMarketItem[i + 1].owner == address(this)) { uint256 currentId = i + 1; MarketItem storage currentItem = idToMarketItem[currentId]; items[currentIndex] = currentItem; currentIndex += 1; } } return items; } /\* Returns only items that a user has purchased \*/ function fetchMyNFTs() public view returns (MarketItem[] memory) { uint256 totalItemCount = \_tokenIds.current(); uint256 itemCount = 0; uint256 currentIndex = 0; for (uint256 i = 0; i < totalItemCount; i++) { if (idToMarketItem[i + 1].owner == msg.sender) { itemCount += 1; } } MarketItem[] memory items = new MarketItem[](itemCount); for (uint256 i = 0; i < totalItemCount; i++) { if (idToMarketItem[i + 1].owner == msg.sender) { uint256 currentId = i + 1; MarketItem storage currentItem = idToMarketItem[currentId]; items[currentIndex] = currentItem; currentIndex += 1; } } return items; } /\* Returns only items a user has listed \*/ function fetchItemsListed() public view returns (MarketItem[] memory) { uint256 totalItemCount = \_tokenIds.current(); uint256 itemCount = 0; uint256 currentIndex = 0; for (uint256 i = 0; i < totalItemCount; i++) { if (idToMarketItem[i + 1].seller == msg.sender) { itemCount += 1; } } MarketItem[] memory items = new MarketItem[](itemCount); for (uint256 i = 0; i < totalItemCount; i++) { if (idToMarketItem[i + 1].seller == msg.sender) { uint256 currentId = i + 1; MarketItem storage currentItem = idToMarketItem[currentId]; items[currentIndex] = currentItem; currentIndex += 1; } } return items; }}  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]

3 **Answer**

**A1:**The problem is that because of the way you are minting, you don't own the token, the contract does.  
  
Since you don't own the token, it doesn't appear in marketplaces and you can't call approve or transfer because you are not the owner of the token.  
  
Here is the culprit:  
  
\_mint(address(this), newItemId, amount, "");  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
You are minting to address(this) which is the address of the contract itself.You'll need a way to send the token from the contract to whomever you want using a custom function, or the probably better solution, you can just mint to the address calling your mintToken function by doing:  
  
\_mint(msg.sender, newItemId, amount, "");  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
Best of luck!!

**A2:**When You call safeTransferFrom, require statement is not passing  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
 function safeTransferFrom(address from,address to,uint256 id,uint256 amount,bytes memory data ) public virtual override { require( from == \_msgSender() || isApprovedForAll(from, \_msgSender()), "ERC1155: caller is not owner nor approved" ); \_safeTransferFrom(from, to, id, amount, data); }  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
Most likely you are transferring a token from the account that does not own the token,

**A3:**You are using safeTransferFrom which is a public function. Try using \_safeTransferFrom.You should add a custom function users can call, and internally use \_safeTransferFrom