# 583.https://stackoverflow.com/questions/69623419/how-to-set-different-whitelist-price-tiers-in-a-erc-721-contract

**T:**How to set different whitelist price tiers in a ERC-721 contract?

**Q:**Hey all thanks so much for this forum have been learning a lot from your insights and feedbacks to each post. I just started doing solidity development and want to say this group is my go-to source for anything solidity related!  
  
I have a smart contract question hope some of you can help me:  
  
I want three groups of ERC-721 token holders to have three different whitelist prices.  
  
the first group (500 holders) have free mint,the second group (500 holders) can mint at 0.003eth per mintthe third group (3000 holders) can mint at 0.004eth per mint.  
  
This new contract have a total of 4000 ERC-721 tokens, how can I code this smart contract without messing up the token mint ID? So far through my experiments the second group may claim the first group's tokens.  
  
Please see my code here, thanks a lot for your help!  
  
//SPDX-License-Identifier: MITpragma solidity ^0.8.0;interface BullInterface { function ownerOf(uint256 tokenId) external view returns (address owner); function balanceOf(address owner) external view returns (uint256 balance); function tokenOfOwnerByIndex(address owner, uint256 index) external view returns (uint256 tokenId);}interface BearInterface { function ownerOf(uint256 tokenId) external view returns (address owner); function balanceOf(address owner) external view returns (uint256 balance); function tokenOfOwnerByIndex(address owner, uint256 index) external view returns (uint256 tokenId);}interface DragonInterface { function ownerOf(uint256 tokenId) external view returns (address owner); function balanceOf(address owner) external view returns (uint256 balance); function tokenOfOwnerByIndex(address owner, uint256 index) external view returns (uint256 tokenId);}contract Environment is ERC721Enumerable, Ownable { using Strings for uint256; string public baseExtension = ".json"; uint256 public cost = 0.005 ether; uint256 public maxBullSupply = 300; uint256 public maxBearDragonSupply = 1200; uint256 public maxEnvironmentSupply = 1500; uint256 public maxMintAmount = 10; bool public paused = false; string public baseURI = "https://ipfs.io/ipfs/QmeSjSinHpPnmXmspMjwiXyN6zS4E9zccariGR3jxcaWtq/"; address public BullAddress = 0x09646c5c1e42ede848A57d6542382C32f2877164; BullInterface BullContract = BullInterface(BullAddress); uint public BullOwnersSupplyMinted = 0; uint public BearDragonSupplyMinted = 0; address public BearAddress = 0x5258589F6ffb08830C5aE43B57aAC01804692223; BearInterface BearContract = BearInterface(BearAddress); address public DragonAddress = 0x2900A5a17Ce7241660dF9f1FD2D24f126Cde69Db; DragonInterface DragonContract = DragonInterface(DragonAddress); constructor( ) ERC721("Environment", "ENVIR") {} // internal function \_baseURI() internal view virtual override returns (string memory) { return baseURI; } function BullFreeMint(uint bullId) public payable { require(bullId > 0 && bullId <= 300, "Token ID invalid"); require(BullContract.ownerOf(bullId) == msg.sender, "Not the owner of this Bull"); \_safeMint(msg.sender, bullId); } function BullMultiFreeMint(uint256[] memory bullIds) public payable { for (uint256 i = 0; i < bullIds.length; i++) { require(BullContract.ownerOf(bullIds[i]) == msg.sender, "Not the owner of these Bulls"); \_safeMint(\_msgSender(), bullIds[i]); } } function BearWhitelistMint(uint bearId, uint \_mintAmount) public payable { require(BearContract.ownerOf(bearId) == msg.sender, "Not the owner of this Bear"); require(msg.value >= 0.003 ether \* \_mintAmount); require(!paused); require(\_mintAmount > 0); require(\_mintAmount <= maxMintAmount); require(totalSupply() + \_mintAmount <= maxEnvironmentSupply); require(BearDragonSupplyMinted + \_mintAmount <= maxEnvironmentSupply, "No more BearDragon supply left"); BearDragonSupplyMinted = BearDragonSupplyMinted + \_mintAmount; for (uint256 i = 1; i <= \_mintAmount; i++) { \_safeMint(msg.sender, totalSupply() + i); } } function DragonWhitelistMint(uint dragonId, uint \_mintAmount) public payable { require(DragonContract.ownerOf(dragonId) == msg.sender, "Not the owner of this Dragon"); require(msg.value >= 0.004 ether \* \_mintAmount); require(!paused); require(\_mintAmount > 0); require(\_mintAmount <= maxMintAmount); require(totalSupply() + \_mintAmount <= maxEnvironmentSupply); require(BearDragonSupplyMinted + \_mintAmount <= maxEnvironmentSupply, "No more BearDragon supply left"); BearDragonSupplyMinted = BearDragonSupplyMinted + \_mintAmount; for (uint256 i = 1; i <= \_mintAmount; i++) { \_safeMint(msg.sender, totalSupply() + i); } } function tokenURI(uint256 tokenId) public view virtual override returns (string memory) { require( \_exists(tokenId), "ERC721Metadata: URI query for nonexistent token" ); string memory currentBaseURI = \_baseURI(); return bytes(currentBaseURI).length > 0 ? string(abi.encodePacked(currentBaseURI, tokenId.toString(), baseExtension)) : ""; } //only owner function withdraw() public payable onlyOwner { require(payable(msg.sender).send(address(this).balance)); }}  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]

0 **Answer**