# 3.https://stackoverflow.com/questions/70139247/solidity-0-8-0-override-virtual-typeerror-derived-contract-must-override-fu

**T:**Solidity ^0.8.0. override virtual. "TypeError: Derived contract must override function .." but it already has the virtual override keywords

**Q:**Context: Solidity ^0.8.0  
  
Purpose: implementing a simple NFT contract  
  
Problem: the error throws i have to specify virtual/override keywords, but i've already done it. ERC721URIStorage inherits from ERC721  
  
Verbose1:  
  
TypeError: Derived contract must override function "\_burn". Two or more base classes define function with same name and parameter types.[...]TypeError: Derived contract must override function "tokenURI". Two or more base classes define function with same name and parameter types. --> contracts/resp.sol:1185:1:  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
Verbose2:  
  
TypeError: Derived contract must override function "\_burn". Two or more base classes define function with same name and parameter types. --> contracts/resp.sol:1185:1: |1185 | contract MyNFT is ERC721, Ownable, ERC721URIStorage { | ^ (Relevant source part starts here and spans across multiple lines).Note: Definition in "ERC721": --> contracts/resp.sol:883:5: |883 | function \_burn(uint256 tokenId) internal virtual { | ^ (Relevant source part starts here and spans across multiple lines).Note: Definition in "ERC721URIStorage": --> contracts/resp.sol:1051:5: |1051 | function \_burn(uint256 tokenId) internal virtual override { | ^ (Relevant source part starts here and spans across multiple lines).TypeError: Derived contract must override function "tokenURI". Two or more base classes define function with same name and parameter types. --> contracts/resp.sol:1185:1: |1185 | contract MyNFT is ERC721, Ownable, ERC721URIStorage { | ^ (Relevant source part starts here and spans across multiple lines).Note: Definition in "ERC721": --> contracts/resp.sol:671:5: |671 | function tokenURI(uint256 tokenId) public view virtual override returns (string memory) { | ^ (Relevant source part starts here and spans across multiple lines).Note: Definition in "ERC721URIStorage": --> contracts/resp.sol:1011:5: |1011 | function tokenURI(uint256 tokenId) public view virtual override returns (string memory) { | ^ (Relevant source part starts here and spans across multiple lines).Warning: Visibility for constructor is ignored. If you want the contract to be non-deployable, making it "abstract" is sufficient. --> contracts/resp.sol:1190:3: |1190 | constructor() public ERC721("MyNFT","NFT") {} | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^Error HH600: Compilation failedFor more info go to https://hardhat.org/HH600 or run Hardhat with --show-stack-traces  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
Source code (flattened)i had to truncate some contracts into [...] bc it takes up too much space, more than the limit here, but are the standard ones from openzeppelin.i only left expanded the contracts in conflict (ERC721, ERC721URIStorage, MyNFT)  
  
// Sources flattened with hardhat v2.6.8 https://hardhat.org// File @openzeppelin/contracts/utils/introspection/IERC165.sol@v4.3.3// SPDX-License-Identifier: MITpragma solidity ^0.8.0;interface IERC165 { [...]}// File @openzeppelin/contracts/token/ERC721/IERC721.sol@v4.3.3pragma solidity ^0.8.0;interface IERC721 is IERC165 { [...]}// File @openzeppelin/contracts/token/ERC721/IERC721Receiver.sol@v4.3.3pragma solidity ^0.8.0;/\*\* \* @title ERC721 token receiver interface \* @dev Interface for any contract that wants to support safeTransfers \* from ERC721 asset contracts. \*/interface IERC721Receiver { [...]}// File @openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol@v4.3.3pragma solidity ^0.8.0;/\*\* \* @title ERC-721 Non-Fungible Token Standard, optional metadata extension \* @dev See https://eips.ethereum.org/EIPS/eip-721 \*/interface IERC721Metadata is IERC721 { [...]}// File @openzeppelin/contracts/utils/Address.sol@v4.3.3pragma solidity ^0.8.0;/\*\* \* @dev Collection of functions related to the address type \*/library Address { [...]}// File @openzeppelin/contracts/utils/Context.sol@v4.3.3pragma solidity ^0.8.0;abstract contract Context { [...]}// File @openzeppelin/contracts/utils/Strings.sol@v4.3.3pragma solidity ^0.8.0;/\*\* \* @dev String operations. \*/library Strings { [...]}// File @openzeppelin/contracts/utils/introspection/ERC165.sol@v4.3.3pragma solidity ^0.8.0;abstract contract ERC165 is IERC165 { [...]}// File @openzeppelin/contracts/token/ERC721/ERC721.sol@v4.3.3pragma solidity ^0.8.0;/\*\* \* @dev Implementation of https://eips.ethereum.org/EIPS/eip-721[ERC721] Non-Fungible Token Standard, including \* the Metadata extension, but not including the Enumerable extension, which is available separately as \* {ERC721Enumerable}. \*/contract ERC721 is Context, ERC165, IERC721, IERC721Metadata { using Address for address; using Strings for uint256; // Token name string private \_name; // Token symbol string private \_symbol; // Mapping from token ID to owner address mapping(uint256 => address) private \_owners; // Mapping owner address to token count mapping(address => uint256) private \_balances; // Mapping from token ID to approved address mapping(uint256 => address) private \_tokenApprovals; // Mapping from owner to operator approvals mapping(address => mapping(address => bool)) private \_operatorApprovals; /\*\* \* @dev Initializes the contract by setting a `name` and a `symbol` to the token collection. \*/ constructor(string memory name\_, string memory symbol\_) { \_name = name\_; \_symbol = symbol\_; } /\*\* \* @dev See {IERC165-supportsInterface}. \*/ function supportsInterface(bytes4 interfaceId) public view virtual override(ERC165, IERC165) returns (bool) { return interfaceId == type(IERC721).interfaceId || interfaceId == type(IERC721Metadata).interfaceId || super.supportsInterface(interfaceId); } /\*\* \* @dev See {IERC721-balanceOf}. \*/ function balanceOf(address owner) public view virtual override returns (uint256) { require(owner != address(0), "ERC721: balance query for the zero address"); return \_balances[owner]; } /\*\* \* @dev See {IERC721-ownerOf}. \*/ function ownerOf(uint256 tokenId) public view virtual override returns (address) { address owner = \_owners[tokenId]; require(owner != address(0), "ERC721: owner query for nonexistent token"); return owner; } /\*\* \* @dev See {IERC721Metadata-name}. \*/ function name() public view virtual override returns (string memory) { return \_name; } /\*\* \* @dev See {IERC721Metadata-symbol}. \*/ function symbol() public view virtual override returns (string memory) { return \_symbol; } /\*\* \* @dev See {IERC721Metadata-tokenURI}. \*/ function tokenURI(uint256 tokenId) public view virtual override returns (string memory) { require(\_exists(tokenId), "ERC721Metadata: URI query for nonexistent token"); string memory baseURI = \_baseURI(); return bytes(baseURI).length > 0 ? string(abi.encodePacked(baseURI, tokenId.toString())) : ""; } /\*\* \* @dev Base URI for computing {tokenURI}. If set, the resulting URI for each \* token will be the concatenation of the `baseURI` and the `tokenId`. Empty \* by default, can be overriden in child contracts. \*/ function \_baseURI() internal view virtual returns (string memory) { return ""; } /\*\* \* @dev See {IERC721-approve}. \*/ function approve(address to, uint256 tokenId) public virtual override { address owner = ERC721.ownerOf(tokenId); require(to != owner, "ERC721: approval to current owner"); require( \_msgSender() == owner || isApprovedForAll(owner, \_msgSender()), "ERC721: approve caller is not owner nor approved for all" ); \_approve(to, tokenId); } /\*\* \* @dev See {IERC721-getApproved}. \*/ function getApproved(uint256 tokenId) public view virtual override returns (address) { require(\_exists(tokenId), "ERC721: approved query for nonexistent token"); return \_tokenApprovals[tokenId]; } /\*\* \* @dev See {IERC721-setApprovalForAll}. \*/ function setApprovalForAll(address operator, bool approved) public virtual override { require(operator != \_msgSender(), "ERC721: approve to caller"); \_operatorApprovals[\_msgSender()][operator] = approved; emit ApprovalForAll(\_msgSender(), operator, approved); } /\*\* \* @dev See {IERC721-isApprovedForAll}. \*/ function isApprovedForAll(address owner, address operator) public view virtual override returns (bool) { return \_operatorApprovals[owner][operator]; } /\*\* \* @dev See {IERC721-transferFrom}. \*/ function transferFrom( address from, address to, uint256 tokenId ) public virtual override { //solhint-disable-next-line max-line-length require(\_isApprovedOrOwner(\_msgSender(), tokenId), "ERC721: transfer caller is not owner nor approved"); \_transfer(from, to, tokenId); } /\*\* \* @dev See {IERC721-safeTransferFrom}. \*/ function safeTransferFrom( address from, address to, uint256 tokenId ) public virtual override { safeTransferFrom(from, to, tokenId, ""); } /\*\* \* @dev See {IERC721-safeTransferFrom}. \*/ function safeTransferFrom( address from, address to, uint256 tokenId, bytes memory \_data ) public virtual override { require(\_isApprovedOrOwner(\_msgSender(), tokenId), "ERC721: transfer caller is not owner nor approved"); \_safeTransfer(from, to, tokenId, \_data); } /\*\* \* @dev Safely transfers `tokenId` token from `from` to `to`, checking first that contract recipients \* are aware of the ERC721 protocol to prevent tokens from being forever locked. \* \* `\_data` is additional data, it has no specified format and it is sent in call to `to`. \* \* This internal function is equivalent to {safeTransferFrom}, and can be used to e.g. \* implement alternative mechanisms to perform token transfer, such as signature-based. \* \* Requirements: \* \* - `from` cannot be the zero address. \* - `to` cannot be the zero address. \* - `tokenId` token must exist and be owned by `from`. \* - If `to` refers to a smart contract, it must implement {IERC721Receiver-onERC721Received}, which is called upon a safe transfer. \* \* Emits a {Transfer} event. \*/ function \_safeTransfer( address from, address to, uint256 tokenId, bytes memory \_data ) internal virtual { \_transfer(from, to, tokenId); require(\_checkOnERC721Received(from, to, tokenId, \_data), "ERC721: transfer to non ERC721Receiver implementer"); } /\*\* \* @dev Returns whether `tokenId` exists. \* \* Tokens can be managed by their owner or approved accounts via {approve} or {setApprovalForAll}. \* \* Tokens start existing when they are minted (`\_mint`), \* and stop existing when they are burned (`\_burn`). \*/ function \_exists(uint256 tokenId) internal view virtual returns (bool) { return \_owners[tokenId] != address(0); } /\*\* \* @dev Returns whether `spender` is allowed to manage `tokenId`. \* \* Requirements: \* \* - `tokenId` must exist. \*/ function \_isApprovedOrOwner(address spender, uint256 tokenId) internal view virtual returns (bool) { require(\_exists(tokenId), "ERC721: operator query for nonexistent token"); address owner = ERC721.ownerOf(tokenId); return (spender == owner || getApproved(tokenId) == spender || isApprovedForAll(owner, spender)); } /\*\* \* @dev Safely mints `tokenId` and transfers it to `to`. \* \* Requirements: \* \* - `tokenId` must not exist. \* - If `to` refers to a smart contract, it must implement {IERC721Receiver-onERC721Received}, which is called upon a safe transfer. \* \* Emits a {Transfer} event. \*/ function \_safeMint(address to, uint256 tokenId) internal virtual { \_safeMint(to, tokenId, ""); } /\*\* \* @dev Same as {xref-ERC721-\_safeMint-address-uint256-}[`\_safeMint`], with an additional `data` parameter which is \* forwarded in {IERC721Receiver-onERC721Received} to contract recipients. \*/ function \_safeMint( address to, uint256 tokenId, bytes memory \_data ) internal virtual { \_mint(to, tokenId); require( \_checkOnERC721Received(address(0), to, tokenId, \_data), "ERC721: transfer to non ERC721Receiver implementer" ); } /\*\* \* @dev Mints `tokenId` and transfers it to `to`. \* \* WARNING: Usage of this method is discouraged, use {\_safeMint} whenever possible \* \* Requirements: \* \* - `tokenId` must not exist. \* - `to` cannot be the zero address. \* \* Emits a {Transfer} event. \*/ function \_mint(address to, uint256 tokenId) internal virtual { require(to != address(0), "ERC721: mint to the zero address"); require(!\_exists(tokenId), "ERC721: token already minted"); \_beforeTokenTransfer(address(0), to, tokenId); \_balances[to] += 1; \_owners[tokenId] = to; emit Transfer(address(0), to, tokenId); } /\*\* \* @dev Destroys `tokenId`. \* The approval is cleared when the token is burned. \* \* Requirements: \* \* - `tokenId` must exist. \* \* Emits a {Transfer} event. \*/ function \_burn(uint256 tokenId) internal virtual { address owner = ERC721.ownerOf(tokenId); \_beforeTokenTransfer(owner, address(0), tokenId); // Clear approvals \_approve(address(0), tokenId); \_balances[owner] -= 1; delete \_owners[tokenId]; emit Transfer(owner, address(0), tokenId); } /\*\* \* @dev Transfers `tokenId` from `from` to `to`. \* As opposed to {transferFrom}, this imposes no restrictions on msg.sender. \* \* Requirements: \* \* - `to` cannot be the zero address. \* - `tokenId` token must be owned by `from`. \* \* Emits a {Transfer} event. \*/ function \_transfer( address from, address to, uint256 tokenId ) internal virtual { require(ERC721.ownerOf(tokenId) == from, "ERC721: transfer of token that is not own"); require(to != address(0), "ERC721: transfer to the zero address"); \_beforeTokenTransfer(from, to, tokenId); // Clear approvals from the previous owner \_approve(address(0), tokenId); \_balances[from] -= 1; \_balances[to] += 1; \_owners[tokenId] = to; emit Transfer(from, to, tokenId); } /\*\* \* @dev Approve `to` to operate on `tokenId` \* \* Emits a {Approval} event. \*/ function \_approve(address to, uint256 tokenId) internal virtual { \_tokenApprovals[tokenId] = to; emit Approval(ERC721.ownerOf(tokenId), to, tokenId); } /\*\* \* @dev Internal function to invoke {IERC721Receiver-onERC721Received} on a target address. \* The call is not executed if the target address is not a contract. \* \* @param from address representing the previous owner of the given token ID \* @param to target address that will receive the tokens \* @param tokenId uint256 ID of the token to be transferred \* @param \_data bytes optional data to send along with the call \* @return bool whether the call correctly returned the expected magic value \*/ function \_checkOnERC721Received( address from, address to, uint256 tokenId, bytes memory \_data ) private returns (bool) { if (to.isContract()) { try IERC721Receiver(to).onERC721Received(\_msgSender(), from, tokenId, \_data) returns (bytes4 retval) { return retval == IERC721Receiver.onERC721Received.selector; } catch (bytes memory reason) { if (reason.length == 0) { revert("ERC721: transfer to non ERC721Receiver implementer"); } else { assembly { revert(add(32, reason), mload(reason)) } } } } else { return true; } } /\*\* \* @dev Hook that is called before any token transfer. This includes minting \* and burning. \* \* Calling conditions: \* \* - When `from` and `to` are both non-zero, ``from``'s `tokenId` will be \* transferred to `to`. \* - When `from` is zero, `tokenId` will be minted for `to`. \* - When `to` is zero, ``from``'s `tokenId` will be burned. \* - `from` and `to` are never both zero. \* \* To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks]. \*/ function \_beforeTokenTransfer( address from, address to, uint256 tokenId ) internal virtual {}}// File @openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol@v4.3.3pragma solidity ^0.8.0;/\*\* \* @dev ERC721 token with storage based token URI management. \*/abstract contract ERC721URIStorage is ERC721 { using Strings for uint256; // Optional mapping for token URIs mapping(uint256 => string) private \_tokenURIs; /\*\* \* @dev See {IERC721Metadata-tokenURI}. \*/ function tokenURI(uint256 tokenId) public view virtual override returns (string memory) { require(\_exists(tokenId), "ERC721URIStorage: URI query for nonexistent token"); string memory \_tokenURI = \_tokenURIs[tokenId]; string memory base = \_baseURI(); // If there is no base URI, return the token URI. if (bytes(base).length == 0) { return \_tokenURI; } // If both are set, concatenate the baseURI and tokenURI (via abi.encodePacked). if (bytes(\_tokenURI).length > 0) { return string(abi.encodePacked(base, \_tokenURI)); } return super.tokenURI(tokenId); } /\*\* \* @dev Sets `\_tokenURI` as the tokenURI of `tokenId`. \* \* Requirements: \* \* - `tokenId` must exist. \*/ function \_setTokenURI(uint256 tokenId, string memory \_tokenURI) internal virtual { require(\_exists(tokenId), "ERC721URIStorage: URI set of nonexistent token"); \_tokenURIs[tokenId] = \_tokenURI; } /\*\* \* @dev Destroys `tokenId`. \* The approval is cleared when the token is burned. \* \* Requirements: \* \* - `tokenId` must exist. \* \* Emits a {Transfer} event. \*/ function \_burn(uint256 tokenId) internal virtual override { super.\_burn(tokenId); if (bytes(\_tokenURIs[tokenId]).length != 0) { delete \_tokenURIs[tokenId]; } }}// File @openzeppelin/contracts/utils/Counters.sol@v4.3.3pragma solidity ^0.8.0;library Counters { [...]}// File @openzeppelin/contracts/access/Ownable.sol@v4.3.3pragma solidity ^0.8.0;abstract contract Ownable is Context { [...]}// File temp/myNFT.sol//Contract based on https://docs.openzeppelin.com/contracts/3.x/erc721pragma solidity ^0.8.0;contract MyNFT is ERC721, Ownable, ERC721URIStorage { using Counters for Counters.Counter; Counters.Counter private \_tokenIds; constructor() public ERC721("MyNFT","NFT") {} function mintNFT(address recipient, string memory tokenURI) public onlyOwner returns (uint256) { \_tokenIds.increment(); uint256 newItemId = \_tokenIds.current(); \_mint(recipient, newItemId); \_setTokenURI(newItemId, tokenURI); return newItemId; }} //endcon  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]

2 **Answer**

**A1:**The MyNFT contract derives from both ERC721 and ERC721URIStorage. But the ERC721URIStorage also derives from the ERC721. Since Solidity doesn't have a dependency injection mechanism, it imports the ERC721 for the second time.  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
This causes the \_burn() and other methods to be redefined without the override keyword.  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
Solution: Remove the ERC721 contract from being a direct parent of MyNFT, so that the ERC721 is only imported once (as a parent of the ERC721URIStorage).  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
contract MyNFT is Ownable, ERC721URIStorage {  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]

**C1:**Thank you Petr, you are absolutely right. I realize also that today, I suppose I had to sleep to see the failure :) so, thank you

**A2:**--> myc:18:5:|18 | function balanceOf(address owner) public returns(uint)| ^ (Relevant source part starts here and spans across multiple lines).