8. **Handling Token Expiry:** Describe how you would handle scenarios where a user attempts to transfer expired tokens.

Ans:

Handling scenarios where a user attempts to transfer expired tokens involves implementing a mechanism in the smart contract to check the expiration status of the tokens before allowing a transfer. Here's an example of how you can handle this situation in a smart contract:

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

import "@openzeppelin/contracts/access/Ownable.sol";

// Custom ERC-20 token contract with expiry date

contract ExpirableERC20Token is ERC20, Ownable {

uint256 public expiryDate;

// Constructor to initialize the token with a given name, symbol, initial supply, and expiry date

constructor(

string memory name,

string memory symbol,

address initialHolder,

uint256 initialSupply,

uint256 \_expiryDate

) ERC20(name, symbol) {

\_mint(initialHolder, initialSupply);

expiryDate = \_expiryDate;

}

// Function to transfer tokens, checking for expiry before allowing the transfer

function transfer(address to, uint256 value) public override returns (bool) {

require(block.timestamp < expiryDate, "Token has expired");

return super.transfer(to, value);

}

// Function to transfer tokens on behalf of someone else, checking for expiry

function transferFrom(address from, address to, uint256 value) public override returns (bool) {

require(block.timestamp < expiryDate, "Token has expired");

return super.transferFrom(from, to, value);

}

// Function to update the expiry date (only callable by the owner)

function updateExpiryDate(uint256 newExpiryDate) external onlyOwner {

expiryDate = newExpiryDate;

}

}