**Creating a Bank**

**Set Solidity Version**

****

* Sets the version of solidity to between 0.4.22 and 0.9.0

**Import Separate Contracts**

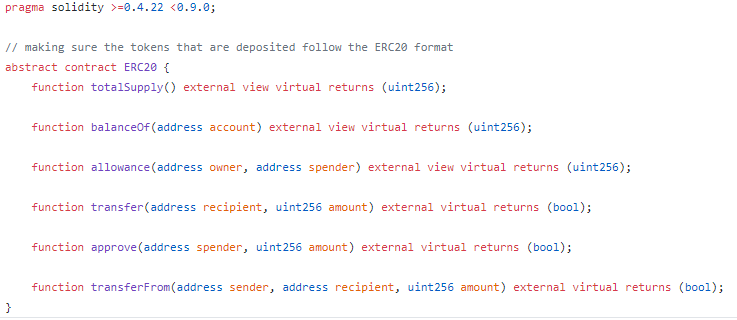
****

**Ownable.sol**

****

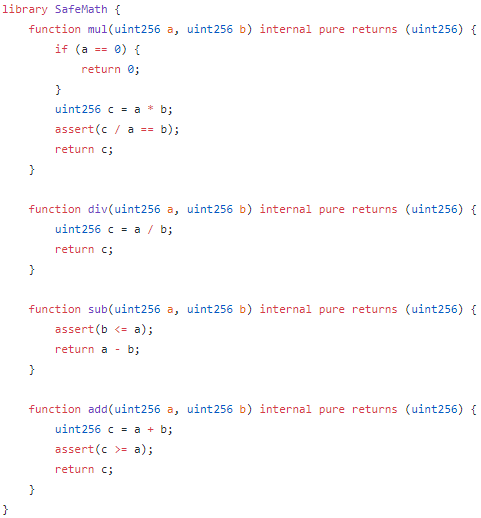
* Handles the management of the owner role, makes sure the person calling certain functions is the owner of the contract

**ERC20.sol**



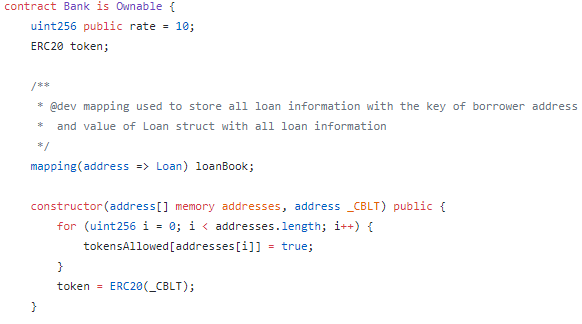
* Sets the basic ERC20 Tokens standards
* In the main Bank.sol contract this helps create a dummy token that we can pass other token addresses and parameters through

**SafeMath**



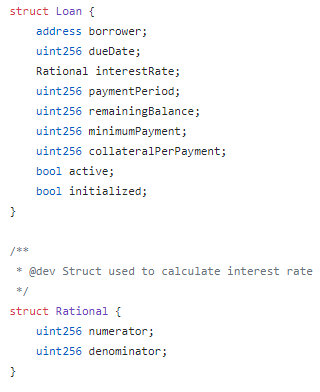
* Helps with the simple math for the contracts, makes sure the calculations are secure

**Starting the contract + Constructor**

****

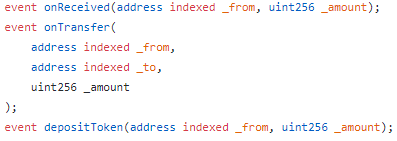
* We use a mapping to attach the address of the user to the loan that they will receive. This information will be put into the variable called loanBook which will record the loans being sent out.
* Constructor Parameters:
  + <address[] memory addresses>: stores the addresses of the allowed tokens in memory and puts them into an array
  + <address \_CBLT>: The address for the CBLT token
* The constructor will use a for loop to assign the allowed tokens from the addresses array
* The CBLT token will be created separately since that token will always be a part of the bank/exchange

**Structs**



* Structs are used to represent a record of an object. In this case we will use structs to hold information that we need for the loans and calculating the interest rate.
* In the <Loan> struct, key information like the borrower’s address, the dueDate, and the collateral is held. This information can either be called later on when the user needs to view it, or can be changed if something relating to the loan is changed
* The <Rational> struct holds the info for the numerator and denominator. This makes it easier to calculate the interest rate which will be used several times throughout this contract.

**Events**

****

* onReceived handles any donations to the treasury.
  + The parameters are the address of the sender, and the amount of currency being sent.
* onTransfer and depositToken handle transfers and deposits
  + The parameters are the addresses of the sender, receiver, and the amount being sent
* These events will be called later on in the contract whenever there is anything to do with donations, transfers, or receiving.

**Mappings**

****

* Sets the total balance of the owner, the list of allowed tokens, the supply of said tokens, and the current balance of ether
* Mappings attach the addresses of the user to these specified values which can be called later on in thee contract

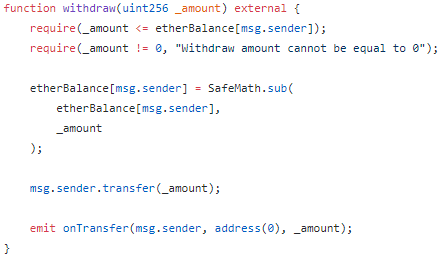
**Functions**

**withdrawTokens**

****

* Parameters:
  + <address \_tokenAddress>: address of the specified token that is being withdrawn
  + <uint256 \_amount>: The number amount that is being withdrawn
* This function allows the user to withdraw tokens assuming that they meet certain requirements:
  + Is the token supported? Does the user have enough in their balance? Is the number 0? The require commands will make sure that these prerequisite are met before the transaction goes through
  + The supported tokens were added in earlier on in the contract in the constructor.
* A dummy token is also created so that the parameters of the supported tokens in the treasury can pass through it.
  + If the user does not meet one of the requirements it will return “Transfer not complete”
* <emit onTransfer> calls the onTransfer event and actually send the tokens

**withdraw**



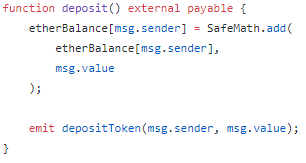
* Very similar to withdrawToken, except for ether so no dummy token needs to be created
* Just like the previous function this one also uses <require> to establish certain prerequisites before the transaction can go through.

**depositTokens**



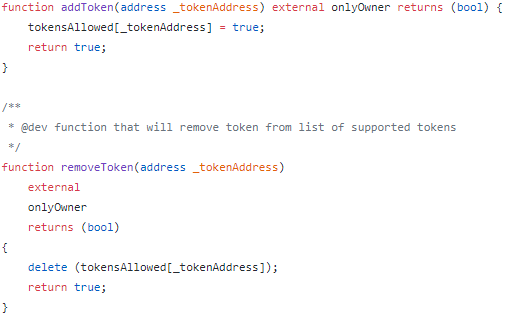
* Parameters:
  + <address \_tokenAddress>: address of the specified token that is being deposited
  + <uint256 \_amount>: The number amount that is being deposited
* This function allows the user to deposit tokens assuming that the token is supported by the treasury. (Very similar to the <withdrawTokens> function)
  + The supported tokens were added in earlier on in the contract in the constructor.
* A dummy token is also created so that the parameters of the supported tokens in the treasury can pass through it.
  + If the user does not meet one of the requirements it will return “Transfer not complete”
* <emit onTransfer> calls the depositToken event and actually send the tokens

**deposit**

****

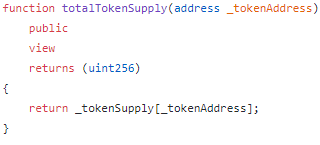
* A basic deposit function used for ether, similar to the withdraw function
* Since ether is an established currency we can forgo the creation of the dummy token that we use for out custom tokens

**addToken/removeToken**



* Parameters:
  + <address \_tokenAddress>: The address of the specified token
* First checks to see if the person who called the function has the role of owner
  + The owner role is created and specified in <Ownable.sol>. Only the owner can make changes as to what tokens are supported by the treasury.
* If the user is the owner, these functions will allow for the addition or removal of certain tokens from the treasury.

**totalTokenSupply**

****

* Returns the total number of a particular token by checking the balance of the contract address.