**bitcci Equity Token Smart Contracts.**

**Prerequisites:**

Node.js, and NPM

Truffle CLI v5.1.x

Ganache

Metamask

solc v0.7.1

@truffle/hdwallet-provider": "^1.0.44"

solhint

**Setup:**

1. Make sure that Ganache is up and running locally at port 8545.
2. Clone The git repository.
3. Install the dependencies using npm install.

If we need to deploy to TestNets such as Ropsten and Rinkeby, create an .env file in this project's root directory and put mnemonic of metamask and infura access token like this:

MNEMONIC='foobar'

INFURA\_ACCESS\_TOKEN='test'

If you're developing locally, you can skip the above step.

**Command to run tests:**  
npm run tests

This command will deploy the contracts first and perform the unit tests.  
  
Note: We can interact with contracts with truffle console as well.

### **How to Deploy on testnet:**

Here's how you can deploy to the Rinkeby testnet:

* Make sure you have [MetaMask](https://metamask.io/) installed.
* On Metamask, select Rinkeby and Import an account using the configured process.env.MNEMONIC and process.env.INFURA\_ACCESS\_TOKEN.
* Copy the account number imported on Metamask to your clipboard,
* Make sure you have enough ether in your account to do the deployment and other transactions. Then, run:

npm run deploy:rinkeby

If all goes well, you should see a successful deployment response.

The contracts are mainly based on the ERC1400 implementation of tenx Contracts which are open-source, scrutinised were audited by chainsecurity.

However the parts which have been taken from tenx are limited to the regulator contract and the token (with no implementation for rewards).   
As the idea is to implement security token with emphasis on restricted transfers on the basis of KYC of investors.   
From feedback by Inacta’s Audit, the contracts are migrated to latest version of solidity(0.7.1) and open zepellin libraries like safemath are updated too.  
Along with that various modifications and enhancements in development have been made.  
Truffle’s hd wallet provider has been updated to latest version as well.

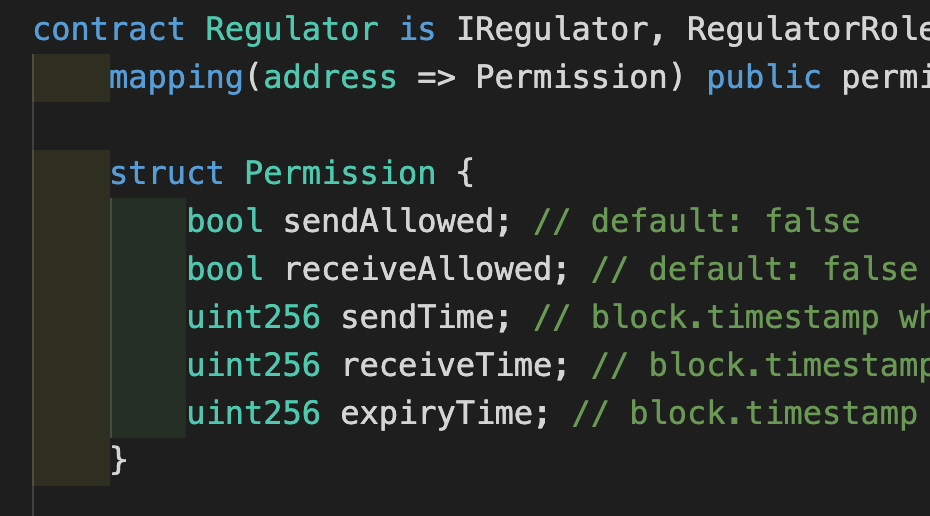
**Regulator Contract:**  
Regulator contracts manages transfer restrictions and implements the IRegulator interface.

Regulator Contract uses Roles library (library/roles.sol) to assign Regulator roles to accounts which will be referred as Regulators.

Regulators are able to modify the whitelists and transfer permissions in Regulator contract.

A key part of a security token is enforcing moderatory restrictions. These business rules are kept out from the token contract.

\* Each address has an associated send, receive, and timelock permissions that either allows or disallows transfers. (both the receiving and the sending parties need to be whitelisted/KYC)



\* Only whitelisted regulator addresses can set permissions.

Additional info related to timelocks:

**There are three time related parameters in struct permission:**

sendTime : block.timestamp when the sale lockup period(if it exists) ends and the investor can freely sell his tokens. default: 0

receiveTime : block.timestamp when purchase lockup period(if it exists) ends and investor can freely purchase tokens from others. default: 0

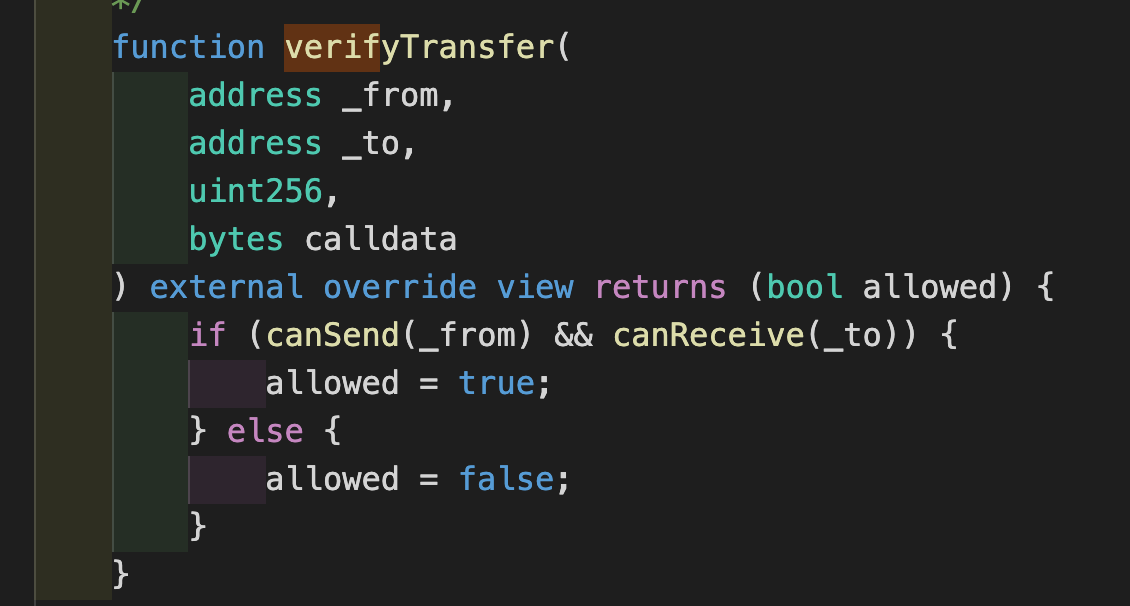
expiryTime : block.timestamp till investors KYC will be validated. After that investor need to do re-KYC. default: 0

The regulator Contract has numerous public view functions to verify whether the issuance/ transfer/ redeem of token is allowed or not beforehand.  
The functions are mentioned here:  
  
verifyIssue();

Returns true if issue is allowed, returns false otherwise

verifyTransfer();

Returns true if transfer is allowed, returns false otherwise.



verifyTransferFrom()

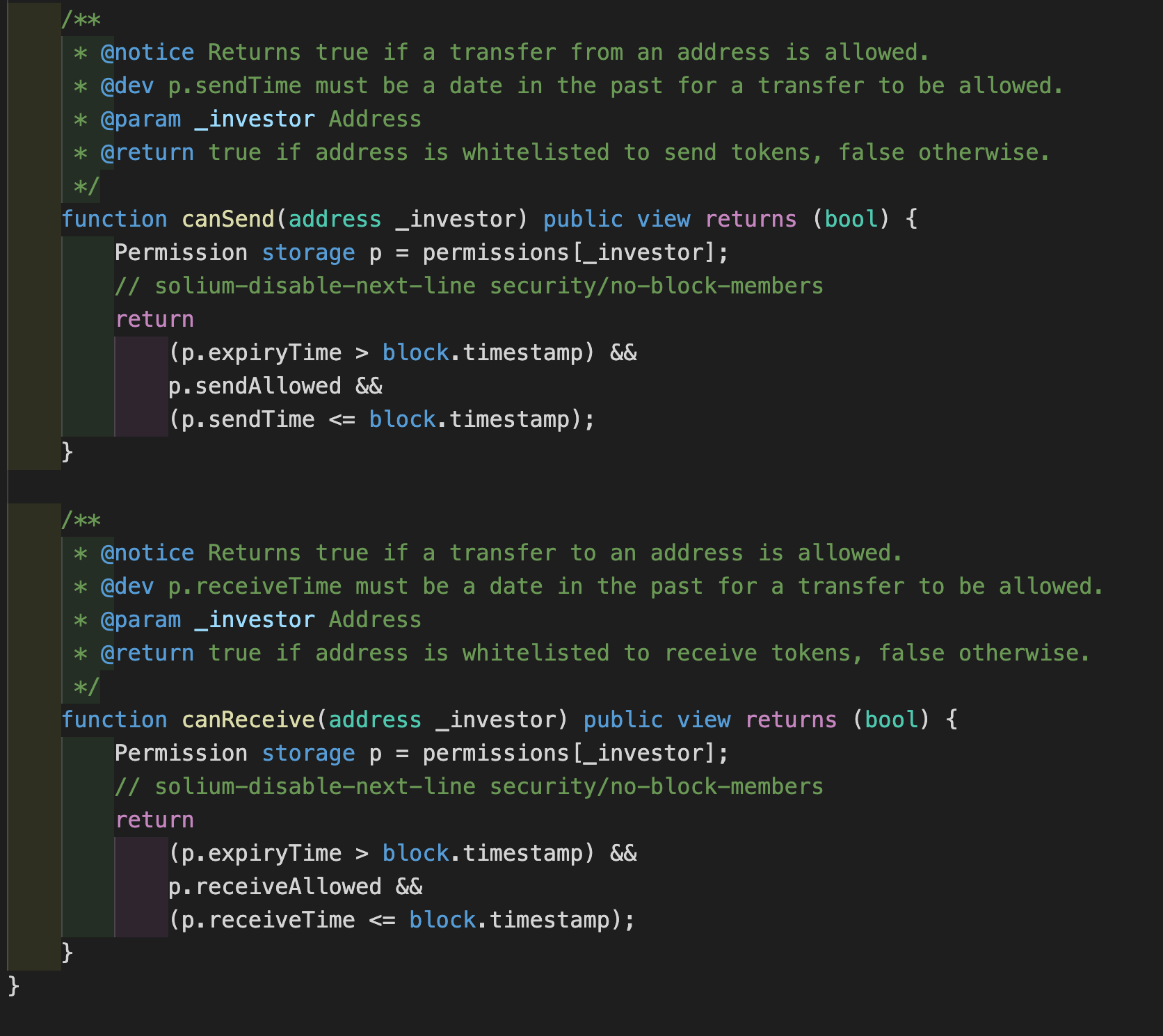
Returns true if transferFrom is allowed, returns false otherwise.

verifyRedeem();

Returns true if redeem(burn) is allowed, returns false otherwise.

Similarly we have verifyRedeemFrom(); verifyControllerTransfer(); verifyControllerRedeem();

The regulator Contract also has canSend and canReceive view functions which returns true if the address passed as parameter is allowed to send/receive tokens. Otherwise it returns false.

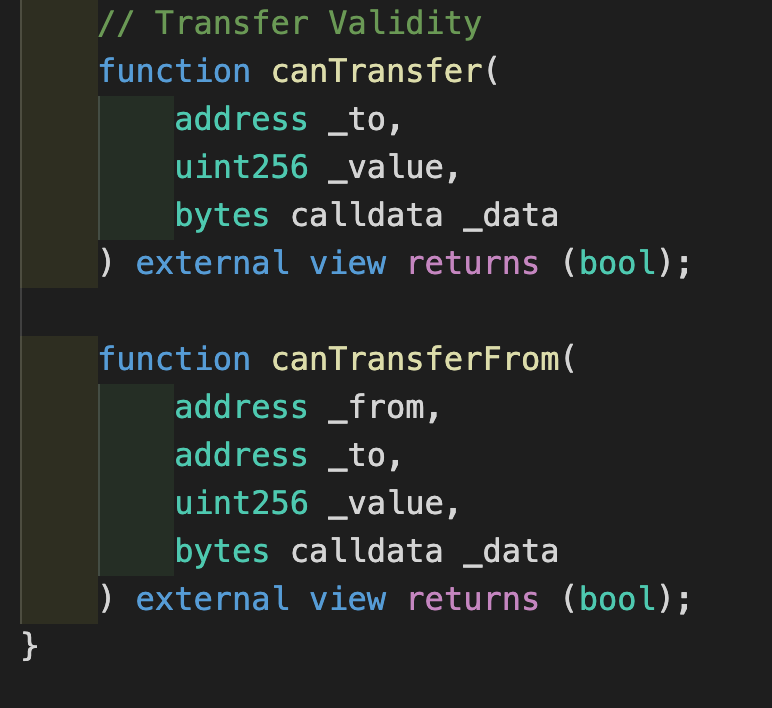


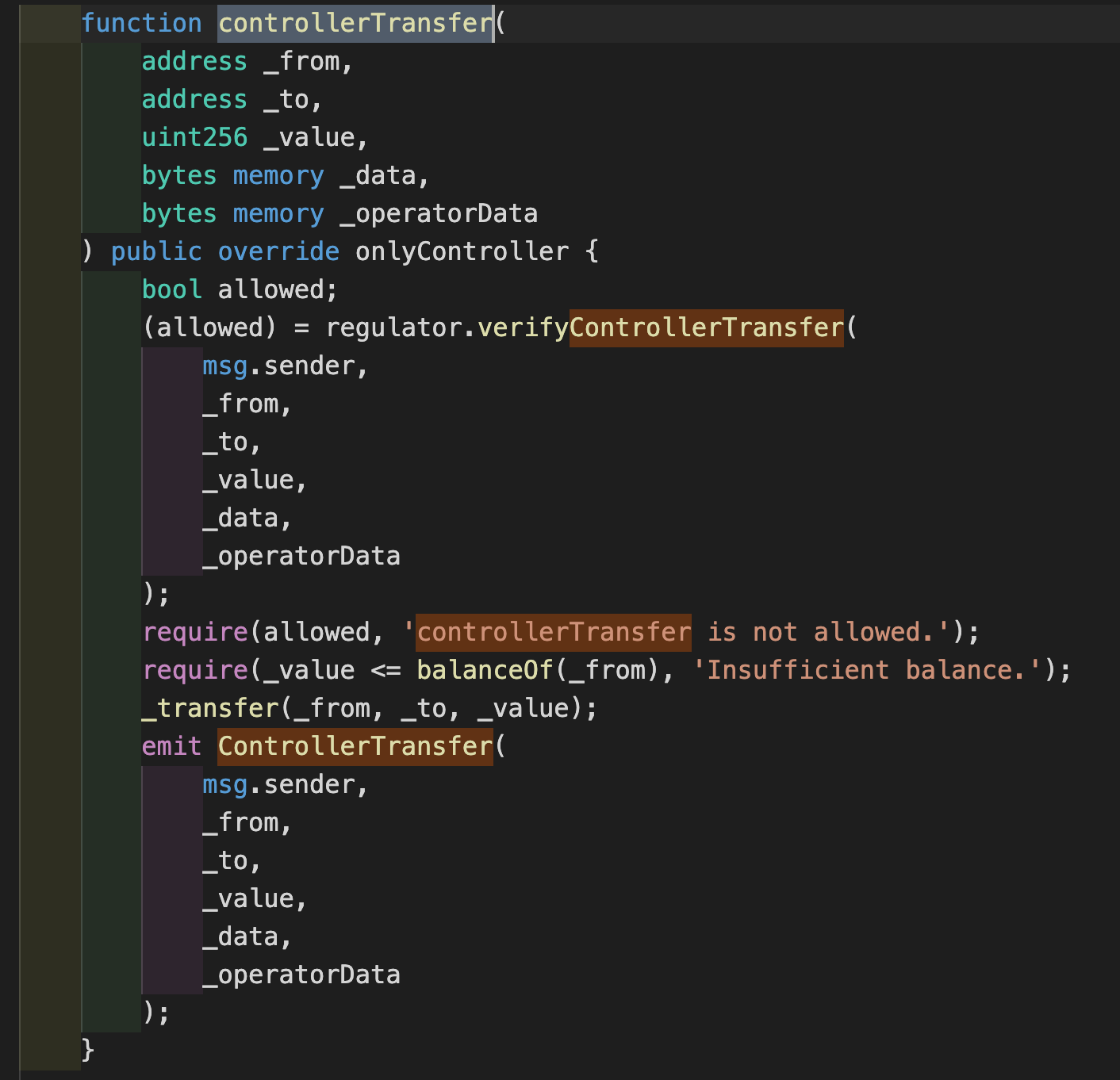
**equity token + regulator Smart Contracts:**

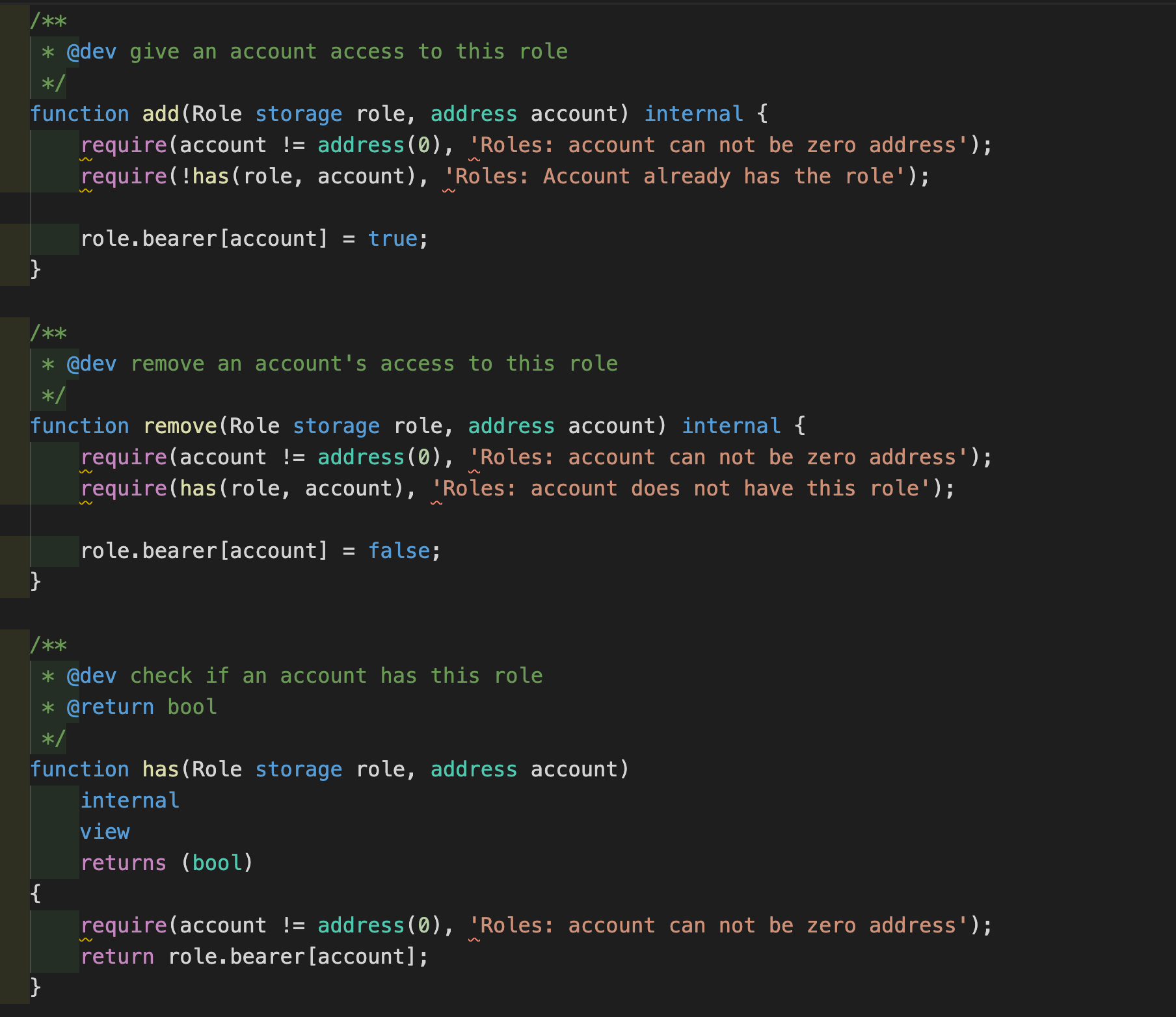
bitcci Equity token is a regulated token which implements ERC1400 Standard.

The following submodules of ERC1400 are implemented:

1. ERC1594 Transfer Restrictions:

The contract implements canTransfer and canTransferFrom functions to check if the token transfer with given parameters is possible or not.  


1. ERC1644 Controller Operations.(Initially the owner retains the ability to perform forced transfers between addresses to reverse fraudulent transactions or the loss of a private key.(controller role can be assigned to trusted party who can perform these controller operations))  
   
2. The contract is using Roles library for managing different roles within the contract, (Issuer Role, Regulator Role, Pauser Role, Controller Role)



Definitions:

Issuer is the account which has authority to issue(mint) new tokens within the cap of Equity token thus increasing the total supply of the token.

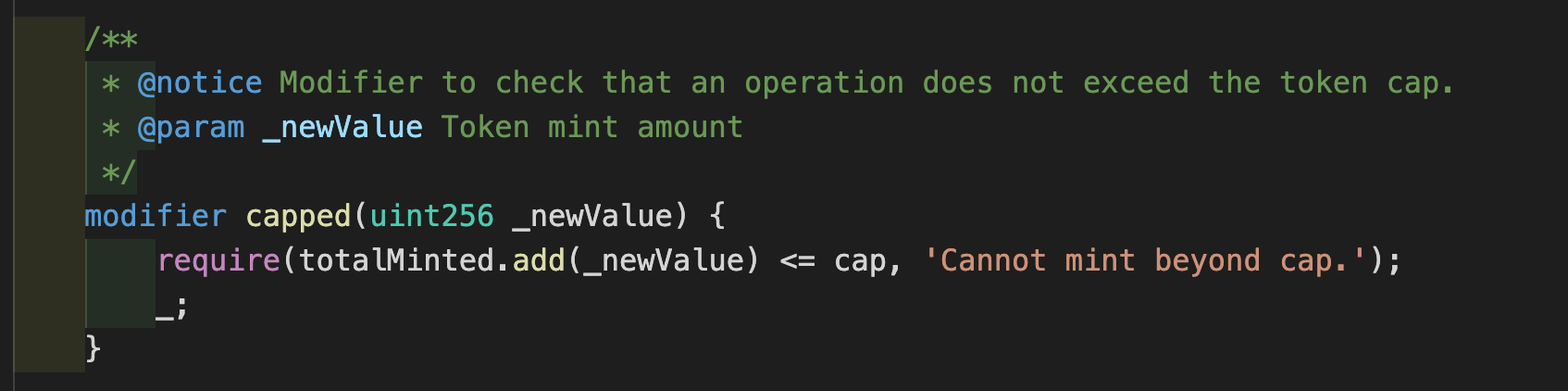
Regulators are able to modify the whitelists and transfer permissions in Regulator contract.

Pauser is the account which has authority to pause/unpause the contract thus putting the contract into a state when no functions can be carried out until unpause function is called by the Pauser account.

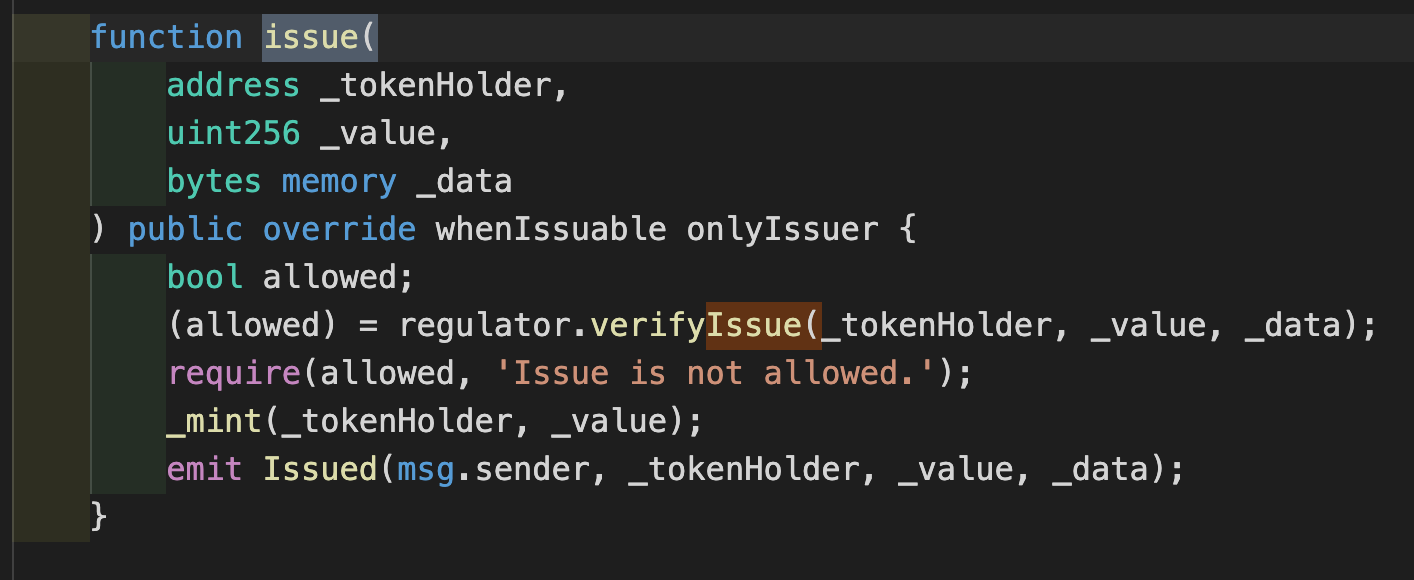
Controller is the account which has the authority to make controller transfers and controller redemptions.  
By default, the owner account has the Issuer, Regulator Controller and Pauser role.

The contract is Ownable and the owner has ability to change the owner, renounce the ownership and execute ownerOnly functions.   
  
Functions with modifier as onlyOwner can only be executed by the owner.

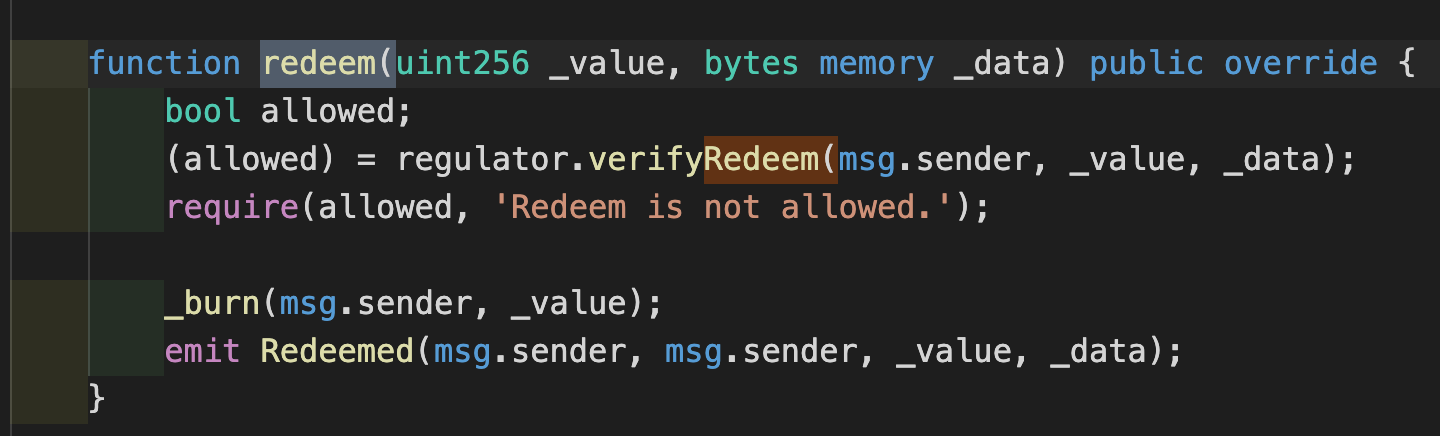
Additional functions:  
1. bitcci Equity token is capped, it means that the total supply of tokens can not exceed the initially mentioned cap.



2. bitcci Equity token is issuable(same as mint): Until the cap is reached issuers can issue more tokens thus increasing the total supply.



3. bitcci Equity token is redeemable (same as burn): The tokens can be burned in order to decrease the total supply.



4. The contract is Pausable.

**Information about Flow of equity token + regulator Smart Contracts:**

Step 1: Deploying the regulator Contract

Step 2: Deployment of equity token contract with following parameters:

1. Regulator Contract(Contract address obtained in step 1)

2. Cap (the maximum number of tokens that can be minted/Issued).

**token sale Smart Contract:**

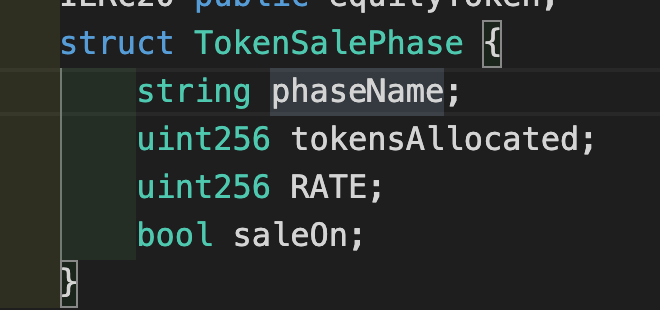
Once the user is KYCed he/she can buy tokens buy sending eth directly to token sale smart contract.

By KYC here we mean that the investor has been give receive permission to receive token with a valid timestamp.

Once KYC is done the user can send ETH to the tokensale contract triggers the buyToken() function.

After taking the feedback from INACTA’s audit the payment method for ownerMultisig has been changed to pull payment method instead of push payment method.  
To add multiple token sale phases the contract maintains an array of tokenSalePhase object.

The token sale object consists of 1. Phase Name 2. Nuber of tokens allocated 3. Rate 4. Bool value stating whether the phase is on or off. (by default it is kept as false)



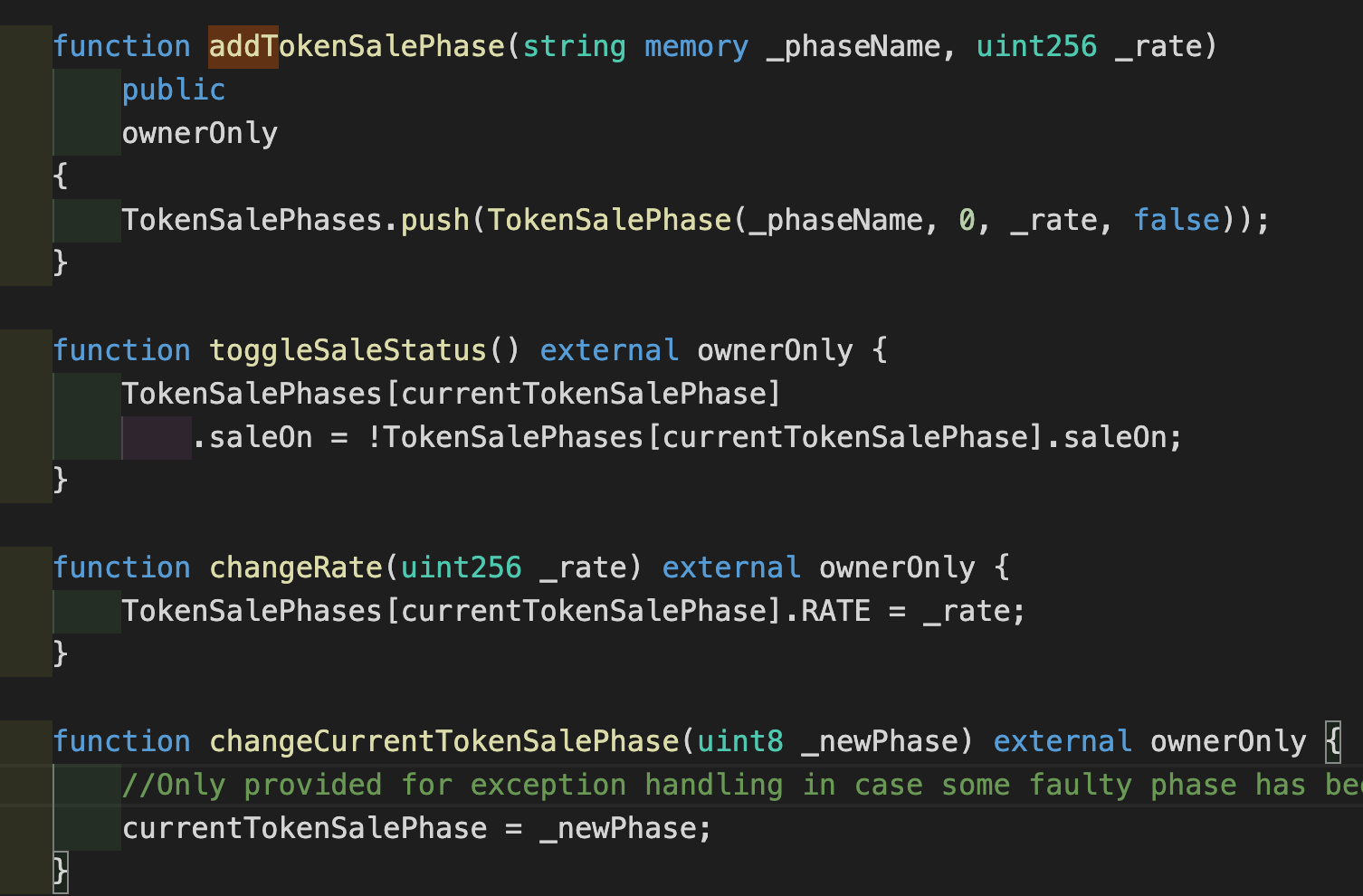
Owner has authority to change the tokenSalePhase.

The contract allows the owner to add more phases of token sale.

Toggle the sale status (on/off)

Change the rate of the token

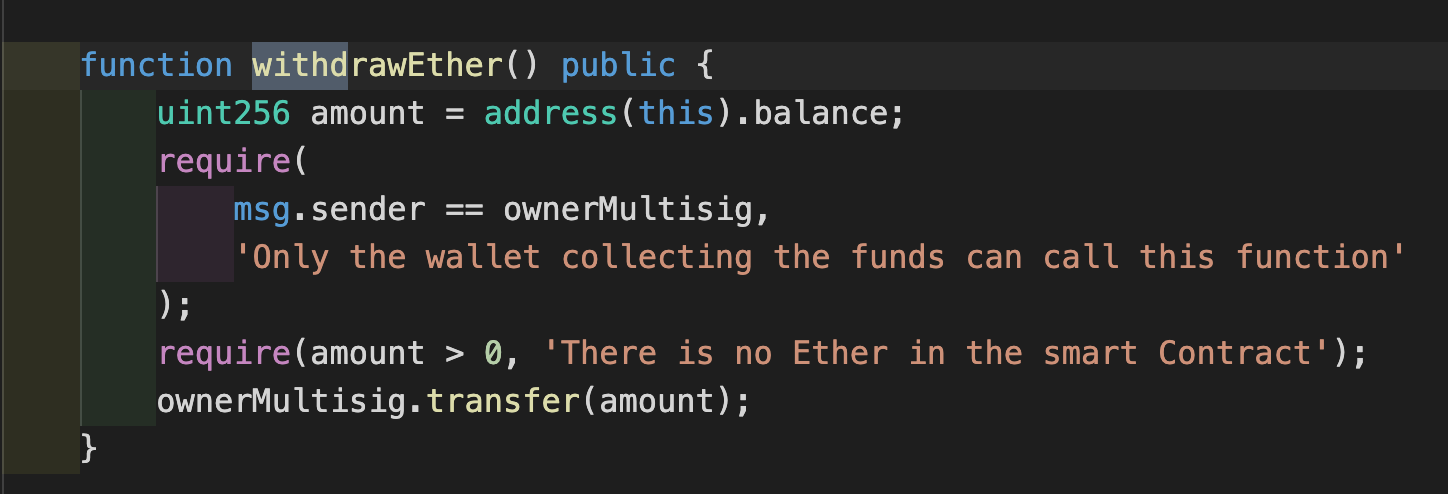
Add new future phases of token sale.



Get the tokens back to owner’s account by Destroy Function.

From the feedback by INACTA we have removed the dangerous SelfDestruct so that investors do not loose their funds by accidently sending eth to destroyed contract.

In order for ownerMultisig wallet to grab the collected funds the contract has withdrawEther() function which transfers the entire ether balance of contract back to ownerMultisig.  
This function has a check which only allows the owner Multisig wallet to call this function.



**Information about Flow of equity token + regulator +Token Sale Smart Contracts:**

Step 1: Deploy the regulator contract (contract A)

Step 2: Deploy the equity token contract with both the parameters(Regulator contract address obtained in step 1 and Cap) (Contract B)

Step 3: Deploy the token sale contract (Contract C) with these parameters:  
 1. equityToken contract address (obtained in step 2 Contract B)

2. ownerMultisig address(address which will collect funds

Step 4: Give receive and send permissions to the contract C using regulator contract (contract A)

Step 5: issue the tokens available for sale to token sale contract

Step 6: execute toggle Sale Status in token sale contract (contract C) to make the token sale live.

Note: The investor should have permission to receive tokens (KYC) in order to get token automatically on sending eth to token sale contract.