Homework 5

Make your contract industry standard using Open Zeppelin libraries

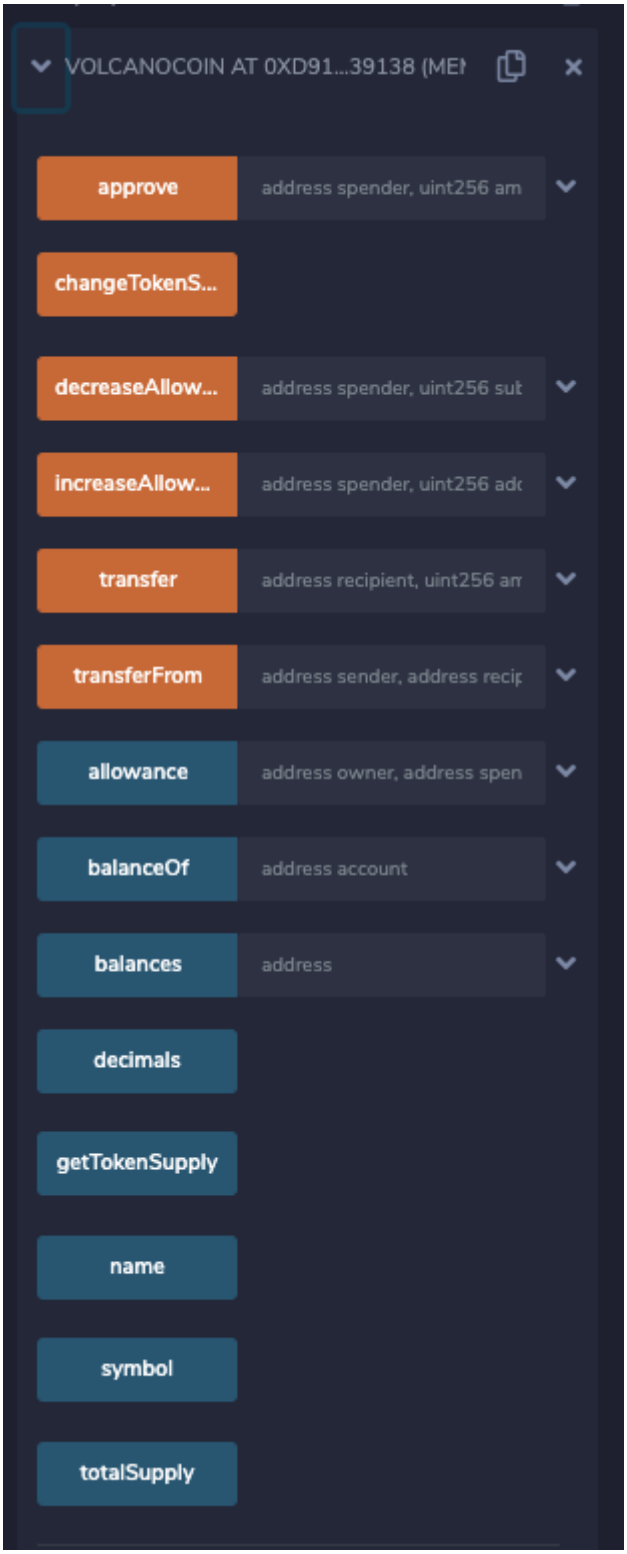
About Open Zeppelin:

Open Zeppelin are well known in the Ethereum community. They provide a set of audited smart contracts and libraries that are a standard in the industry.

Inheriting these contracts will provide a significantly higher degree of security and robustness in your code.

1. An essential function that our contract currently does not have is to transfer tokens from one account to another. The ERC20 standard provides a secure method for doing this. Import the ERC20 standard.
2. Next, inherit the ERC20 contract. You should get a TypeError stating that the contract should be marked as abstract. This is because Abstract contracts are used as base contracts in order for the child contract (VolcanoCoin) to inherit its functions.

   But we want to deploy our contract as an ERC20 token, therefore we need a constructor right after `ERC20` , which requires a string token name and string symbol as an input.
3. Deploy the contract. You should see the additional methods after inheriting the ERC20.
4. Press the drop down arrow under Deployed Contracts. You should be able to see a list of the public functions inherited from ERC20. The orange calls are public write calls (transactions calls which cost gas) and the blue calls are read only calls.



5. You will see that ERC20 provides useful functions, including tracking balances in the contract. We no longer need our `balances` mapping. Remove the mapping and the related code in some of the functions.

6. Open Zeppelin has a contract for `onlyOwner`. Import the contract from the library, then inherit the contract with the `Ownable` keyword.

7. You should see an error by your `modifier` function. You can now remove the function altogether. Notice that the changeTokenSupply, which has the `onlyOwner` keyword is still executable.

Also, you no longer need to define an owner `address`, or have a constructor to set the `owner` as `msg.sender`. The Ownable contract handles this logic for you. Remove the owner variable and its logic in the constructor.

8. ERC20 has an internal function `_mint`. When called, it mints token to the recipient. Create a constructor that calls the `_mint` function inside the constructor. Mint the 10000 token supply to the owner.

9. Make a function that can mint more tokens to the owner.

Adding more functionality

10. Firstly, make your payment record mapping `public` so that view calls can be made to read the contract's data.

11. For the payments record mapping, create a function that takes the sender's address, the receiver's address and amount as an input, then creates a new payment record when a transfer is made and adds the new record to the user's payment record.

12. Recompile and redeploy your contract. Try minting tokens to the owner, and play around with sending tokens from the owner to A, A to B etc. Have a look at the payments record to see how the transaction is recorded in the contract's storage.

13. Deploy your contract to the Rinkeby test network and try exchanging tokens with your team.

Resources

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol
(https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol)