The goal of the task is to understand how object oriented programming works in practice. Along with it, to complete the task you will need to understand the mechanism of ERC20 token standard.

**Info**

Info about ERC20 tokens standard can be previously found [here](https://eips.ethereum.org/EIPS/eip-20).  
Also it is highly recommended to get acquainted with the standard ERC20 token implementation from openzeppelin [here](https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol).

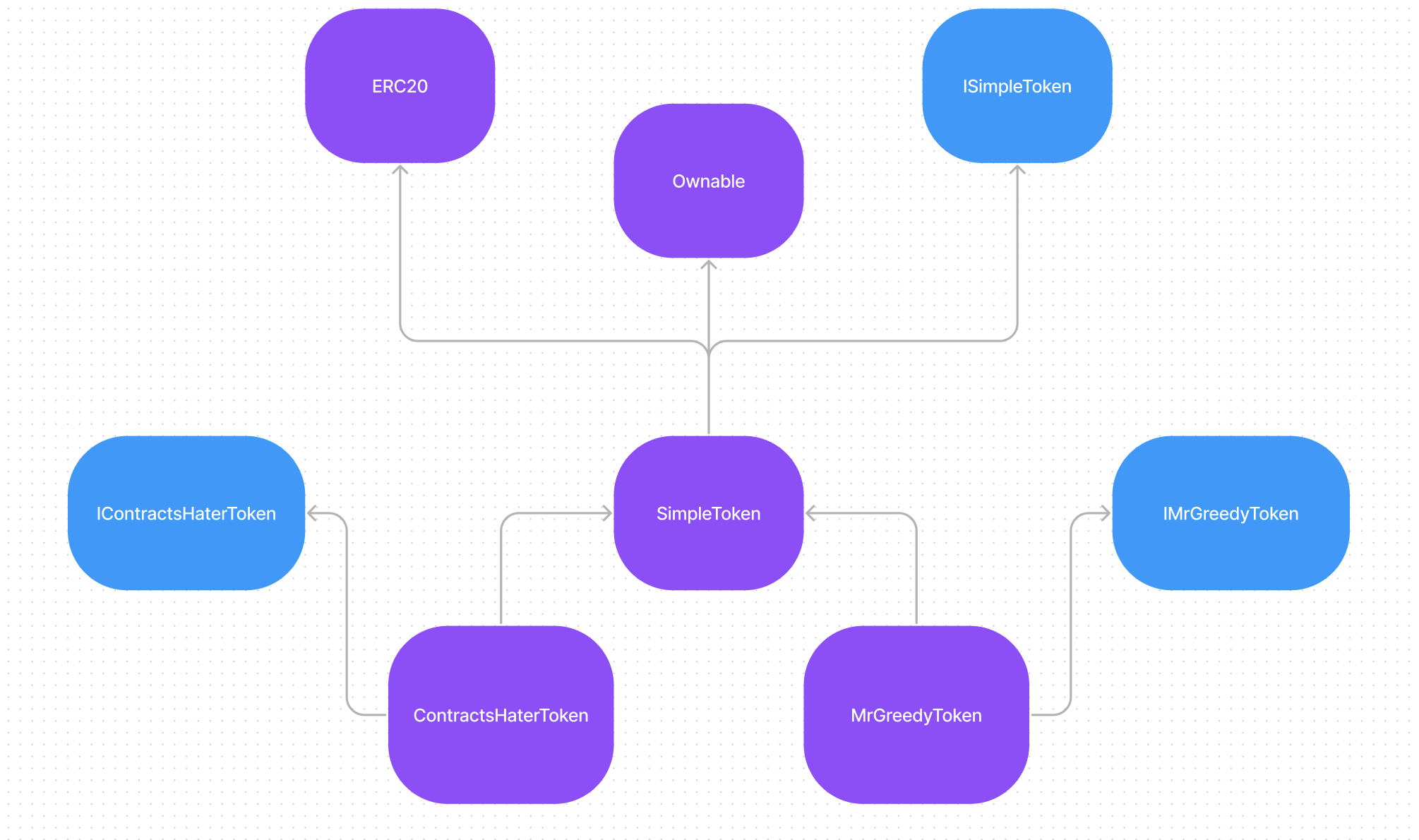
As an advanced topic you can take a look at ERC20 standard extensions in openzeppelin github repo. (Not necessary).

**Task**

You will have to create a batch of SCs each of them must be an ERC20 token contract but all have to be with different functionalities.

1. **SimpleToken** contract.  
     
   Name of the token should be - “SimpleToken”.  
   Symbol of the token - “ST”.  
   Decimals - 18.  
     
   Should be extended by 2 external functions.   
   mint() - function that should allow to mint the number of tokens defined in parameter to the given address. Available to the contract owner only!  
   burn() - should do the opposite, burn provided number of tokens from the sender. Available for anyone.   
   Interface will be provided.
2. **ContractsHaterToken** contract.  
     
   Name of the token should be - “ContractsHaterToken”.  
   Symbol of the token - “CHT”.  
   Decimals - 18.  
     
   Should extend the logic of **SimpleToken** contract, meaning that it also should be mintable/burnable. But along with it should implement some new logic.   
   This contract will have some restrictions applied to its transfer. More precisely it should **revert** any transfer sent to the **smart contract address** but transfer to ordinary addresses should be as usual. Also the logic will have the exclusion from that rule. This token should also have the logic of **whitelisting**. It means that transfer will pass to the contract that is provided in the **whitelist**.  
   About the **whitelist**. It is a simple data structure that stores info about addresses that are allowed to receive transfers. Working with whitelist is easy enough - it will have two functions also provided in the interface.  
   addToWhitelist() - should add a new address to the whitelist.  
   removeFromWhiteList() - should remove the address from the whitelist.  
   Both should be accessible for the contract owner only!  
   To summarize, the logic of the transfer should be next, if address is a SC check the whitelist, if the address in the whitelist - pass the transaction, otherwise - revert. If the address is not SC - just pass.   
     
   Importantly, this restriction should affect only transfer, **mint** and **burn** should be as usual.  
   And one major restriction - internal function transfer must **not** be overridden here!
3. **MrGreedyToken** contract.  
     
   Name of the token should be - “MrGreedyToken”.  
   Symbol of the token - “MRG”.  
   Decimals - 6.  
     
   This token should implement all the logic from **SimpleToken** and also have its modification.  
     
   Modification is next, any transfer should pay a fee to the treasury address. It is always going to be 10 full tokens. If transfer amount is below 10 tokens fee, then everything goes for a fee.  
   As an example, if address A transfers 100 tokens to address B, then B gets 90 tokens and the treasury address gets 10. If address A transfers 5 tokens to B, address B gets 0 and treasury gets 5 tokens.  
     
   Treasury address should be set during deployment and cannot ever be changed. Importantly, it should be public.  
     
   Also, the contract should implement the view function for showing resulting transfer amount. Meaning that the user can pass an amount and the contract will return (amount - 10). Will be provided in the interface.

To summarize you will need to create 3 ERC20 token contracts with different additional logic each.

And the scheme of inheritance should be next  
  


**Validation**

To validate your contracts first of all **Ownership** of all 3 contracts should be transferred to the [**Validator**](https://ropsten.etherscan.io/address/0x1692c54524d8b674c80fCEf5197071235DB59708) contract address. Then all as usual call the function validate with 3 parameters of 3 contracts accordingly: SimpleToken, ContractsHaterToken, MrGreedyToken. Everything should be deployed on a ropsten network.

And don't be scared about such error message during validation, it is totally fine in this particular task.

