Interface -> means external

All the function should be external

No definition only declaration

No state variables and constructor

Interface factory{

methodPrint(){

}

}

Contract MyContract{

somePublicFunc(){

factory(address).methodPrint();

}

Here we are actually using Interface to point to some other contract, by passing the address of that contact which actually has implemented/defined these methods. See below detailed explanation

**Understanding interface:**

18

I believe I got the main issue that was confusing to me.

So if you come from OOP background this what we know about interfaces:

interface IERC20 {

function totalSupply() external view returns (uint256);

}

contract XYZ is IERC20 {

// then implement totalSupply here

function totalSupply() external view returns (uint256) {

// implementiation goes here.

address public add='0x123...4'

}

So at this point you can call XYZ's totalSupply() and you should be fine.

However, there is another way of using interfaces in Solidity. I will take this code from compound protocol as an example (<https://github.com/compound-developers/compound-supply-examples>)

If you see MyContracts.sol, it has the following interface:

interface CEth {

function mint() external payable;

function exchangeRateCurrent() external returns (uint256);

function supplyRatePerBlock() external returns (uint256);

function redeem(uint) external returns (uint);

function redeemUnderlying(uint) external returns (uint);

}

However, there is no place in our contract that uses the keyword **IS** and implements any of the methods. So you might ask how is our interface being used ?

Now let's go back to MyContract contract in MyContracts.sol file and see this code under supplyEthToCompound function:

CEth cToken = CEth(\_cEtherContract);

Here we are providing CEth interface with a contract address of Compound (i.e \_cEtherContract and the contract at that address has a **mint()** function.)

When you call cToken.exchangeRateCurrent(); on the next line, what happens is we are basically calling a function **exchangeRateCurrent** on Compound Contract.

At first it seems like **exchangeRateCurrent** has no implementation in the file we are calling it but the implementation resides at **\_cEtherContract** address.

I hope this clears the confusion especially if you come from traditional OOP background.

Feel free to point out anything that is misleading in my answer.

}

Abstract vs Interface well explained

**Abstract Contract and Interfaces in Solidity**

[#abstractcontract](https://dev.to/t/abstractcontract)[#solidity](https://dev.to/t/solidity)[#interfaces](https://dev.to/t/interfaces)

**Abstract Contract**

An abstract contract is one that cannot be deployed by itself. An abstract contract must be inherited by another contract. An abstract contract is somewhat similar to an interface but there exist some differences between them. An abstract contract can defined functions signature and can also have implementation for some of its functions.

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

abstract contract SayHello {

uint256 public age;

constructor(uint256 \_age ){

age = \_age;

}

function getAge() public virtual view returns (uint256){

return age;

}

function setAge(uint256 \_age) public virtual {}

function makeMeSayHello() public pure returns (string memory)

{

return "Hello";

}

}

The above contract SayHello has four functions of which one of them setAge is a function signature which an inheriting contract can choose to implement. The abstract contract above can not be deployed on its own like a normal contract. It has to be inherited. The contract has a constructor function and you can also defined state variables on an abstract contract like you will do for a normal contract.

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

abstract contract SayHello {

uint256 public age;

constructor(uint256 \_age ){

age = \_age;

}

function getAge() public virtual view returns (uint256){

return age;

}

function setAge(uint256 \_age) public virtual {}

function makeMeSayHello() public pure returns (string memory)

{

return "Hello";

}

}

contract Hello is SayHello {

string public name;

constructor(string memory \_name ,uint256 \_age)

SayHello(\_age) {

name = \_name;

}

function setName(string memory \_name ) public {

name = \_name;

}

function getName() public view returns (string memory){

return name;

}

function getAge() public override virtual view returns

(uint256){

return 67;

}

function setAge(uint256 \_age ) public override virtual {

age = \_age;

}

}

In the solidity contract above, our contract Hello is inheriting the abstract contract SayHello. The abstract contract SayHello has a constructor function which the inheriting contract Hello calls in its own constructor. The Hello contract can choose to override all the functions of the abstract contract with its own implementation but an inheriting contract like the Hello contract must at least implement one function of the abstract contract or it will be marked as abstract.

**Interfaces**

An interface is similar to an abstract contract as it must be inherited by another contract like the abstract contract. An interface functions visibility must be marked as **external**. It cannot have a constructor neither can it declare state variables.

sample interface implementation

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

interface ISayHello {

function getAge() external view returns (uint256);

function setAge(uint256 \_age) external view returns (uint256);

function makeMeSayHello() external view returns (string memory);

}

An interface must not inherit from another contract or interface and an inheriting contract must implement all the functions of that interface.

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

interface ISayHello {

function getAge() external view returns (uint256);

function setAge(uint256 \_age) external ;

function makeMeSayHello() external view returns (string memory);

}

contract HelloAgain is ISayHello {

uint256 public age;

function getAge() public pure override returns (uint256) {

return 33;

}

function setAge(uint256 \_age) public override {

age = \_age;

}

function makeMeSayHello() public pure override returns (string memory) {

return "Hello again";

}

}

The function HelloAgain inherits the interface ISayHello which must override and implement all the functions of the interface. If any of the function is not overridden, the compiler complains and ask you to mark the inheriting contract as abstract.

**Difference between abstract contract and an interface**

* An interface cannot have a constructor while an abstract contract can implement one.
* An interface cannot define state variables but an abstract contract can.
* An inheriting contract must implement all the functions defined in an interface while in an abstract contract the inheriting contract must implement at least one function of the abstract contract.
* An abstract contract can inherit from another contract or abstract contract while an interface cannot inherit from a contract or another interface.(correction: interface can inherit from another interface)

Interfaces and abstract contracts are arsenals in our toolkit that we could use for the development of smart contracts.

Thanks for reading...