Inheritance

Multiple Inheritance

Copy page

Learn how to have a contract inherit from multiple contracts.

Contracts can inherit from more than one contract. In this lesson, we'll explore how multiple inheritance works in Solidity.

Objectives

By the end of this lesson you should be able to:

- Write a smart contract that inherits from multiple contracts

Multiple Inheritance

Continue working with your contracts in `Inheritance.sol` . Add a new contract called `ContractC` with another `whoAmI` function:

Reveal code

Inheriting from Two Contracts

You can inherit from additional contracts by simply adding a comma and that contract’s name after the first. Add inheritance from `ContractC` (an error is expected):

Reveal code

The error is because both `ContractB` and `ContractC` contain a function called `whoAmI` . As a result, the compiler needs instruction on which to use.

```
from solidity:
TypeError: Derived contract must override function "whoAmI". Two or more ba
  --> contracts/Inheritance.sol:21:1:
   |
21 | contract ContractA is ContractB, ContractC {
   | ^ (Relevant source part starts here and spans across multiple lines).
Note: Definition in "ContractC":
 --> contracts/Inheritance.sol:6:5:
  |
6 |     function whoAmI() external pure returns (string memory) {
  |     ^ (Relevant source part starts here and spans across multiple lines
Note: Definition in "ContractB":
  --> contracts/Inheritance.sol:12:5:
   |
12 |     function whoAmI() external pure returns (string memory) {
   |     ^ (Relevant source part starts here and spans across multiple line
```

Using Virtual and Override

One method to resolve this conflict is to use the `virtual` and `override` keywords to enable you to add functionality to choose which to call.

Add the `virtual` keyword to the `whoAmI` function in both `ContractC` and `ContractB`.

They must also be made `public` instead of `external`, because `external` functions cannot be called within the contract.

```
contract ContractC {
    function whoAmI() public virtual pure returns (string memory) {
        return "contract C";
    }
}

contract ContractB {
    function whoAmI() public virtual pure returns (string memory) {
        return "contract B";
    }

    // ... additional code
}
```

Add an `override` function called `whoAmI` to `ContractA`:

```
// Bad code example, do not use
```

You'll get another error, telling you to specify which contracts this function should override.

```
from solidity:
TypeError: Function needs to specify overridden contracts "ContractB" and "
  --> contracts/Inheritance.sol:22:32:
   |
22 |     function whoAmI() public override pure returns (string memory) {
   |                                ^^^^^^^^
```

Add them both:

```
function whoAmI() external override(ContractB, ContractC) pure returns (str
    return ContractB.whoAmI();
}
```

Deploy and test. The c Ask a question... Ctrl+I B".

Changing Types Dynamically

Add an `enum` at the contract level in `ContractA` with members for `None`, `ContractBType`, and `ContractCType`, and an instance of it called `contractType`.

Reveal code

Add a `constructor` to `ContractA` that accepts a `Type` and sets `initialType`.

Reveal code

Update `whoAmI` in `ContractA` to call the appropriate `virtual` function based on its `currentType`.

Reveal code

You'll get errors because the function now reads from state, so it is no longer `pure`. Update it to `view`. You'll also have to update the `whoAmI` `virtual` functions to `view` to match.

Reveal code

Finally, add a function that allows you to switch `currentType`:

Reveal code

Deploy and test. You'll need to use **0**, **1**, and **2** as values to set `contractType`, because Remix won't know about your `enum`.

Final Code

After completing this exercise, you should have something similar to:

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

contract ContractC {
    function whoAmI() public virtual view returns (string memory) {
        return "contract C";
    }
}

contract ContractB {
    function whoAmI() public virtual view returns (string memory) {
        return "contract B";
    }

    function whoAmIInternal() internal pure returns (string memory) {
        return "contract B";
    }
}

contract ContractA is ContractB, ContractC {
    enum Type { None, ContractBType, ContractCType }

    Type contractType;

    constructor (Type _initialType) {
        contractType = _initialType;
    }
    function changeType(Type _newType) external {
        contractType = _newType;
    }

    function whoAmI() public override(ContractB, ContractC) view returns (s
        if(contractType == Type.ContractBType) {
            return ContractB.whoAmI();
        }
        if(contractType == Type.ContractCType) {
            return ContractC.whoAmI();
        }
        return "contract A";
    }

    function whoAmExternal() external pure returns (string memory) {
        return whoAmIInternal();
    }
}
```

Conclusion

In this lesson, you've explored how to use multiple inheritance to import additional functionality into a contract. You've also implemented one approach to resolving name conflicts between those contracts.

Was this page helpful? Yes No Suggest edits Raise issue

Multiple Inheritance

Abstract Contracts

base docs