**Write an article on this topics;**

* **Inheritance**
* **Shadowing Inherited State Variables**
* **Calling Parent Contracts**
* **Visibility**
* **Interface**

1.**//Inheritance //overide //passing parameters**

pragma solidity ^0.8.17;

contract dad{

function getContractName()public pure returns(string memory){

return "Contract dad";

}

}

//over ride

contract mum is dad{

function getContractName(uint) public pure returns (string memory){

return "Contract mum";

}

}

//passing parameters

/\*contract dad{

string public name;

constructor (string memory \_name)public {

name =\_name;

}

}

contract mum is dad{

constructor(string memory \_name) dad(\_name) public{

}

}

2. **Visibility**

// SPDX-License-Identifier: GPL-3.0

//Visibility private,internal public,   and external

pragma solidity ^0.8.17;

contract Base{

    string public base;

    function privateFunc()private pure returns (string memory){

        return "private Function called";

    }

    function testPrivateFunc() public pure returns (string memory){

        return privateFunc();

    }

    function internalFunc() internal pure returns (string memory){

        return "internal function called";

    }

    function testInternalFunc() public pure returns (string memory){

        return internalFunc();

    }

    function publicFunc() public pure returns (string memory){

        return "publicFunc()";

    }

    function testPublicFucn()public pure returns (string memory){

        return publicFunc();

    }

    function externalFunc() external pure returns (string memory){

        return "external fucntion called";

    }

    function testExternalFunc() public pure returns (string memory){

        return externalFunc();

    }

}

contract Child is Base {

    function testinternalFunc() public pure returns (string memory){

        return internalFunc();

    }

}

3.   **Calling Parent Contracts**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

/\* Inheritance tree

A

/ \

B C

\ /

D

\*/

contract A {

// This is called an event. You can emit events from your function

// and they are logged into the transaction log.

// In our case, this will be useful for tracing function calls.

event Log(string message);

function foo() public virtual {

emit Log("A.foo called");

}

function bar() public virtual {

emit Log("A.bar called");

}

}

contract B is A {

function foo() public virtual override {

emit Log("B.foo called");

A.foo();

}

function bar() public virtual override {

emit Log("B.bar called");

super.bar();

}

}

contract C is A {

function foo() public virtual override {

emit Log("C.foo called");

A.foo();

}

function bar() public virtual override {

emit Log("C.bar called");

super.bar();

}

}

contract D is B, C {

// Try:

// - Call D.foo and check the transaction logs.

// Although D inherits A, B and C, it only called C and then A.

// - Call D.bar and check the transaction logs

// D called C, then B, and finally A.

// Although super was called twice (by B and C) it only called A once.

function foo() public override(B, C) {

super.foo();

}

function bar() public override(B, C) {

super.bar();

}

}

**4. Shadowing Inherited State Variables**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

contract dad {

string public name = "Contract dad";

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

return name;

}

}

// Shadowing is disallowed in Solidity 0.6

// This will not compile

// contract mum is dad {

// string public name = "Contract mum";

// }

contract kid is dad {

// This is the correct way to override inherited state variables.

constructor() {

name = "Contract kid";

}

// kid.getName returns "Contract kid"

}

1. **Interface**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

contract Counter {

uint public count;

function increment() external {

count += 1;

}

}

interface ICounter {

function count() external view returns (uint);

function increment() external;

}

contract MyContract {

function incrementCounter(address \_counter) external {

ICounter(\_counter).increment();

}

function getCount(address \_counter) external view returns (uint) {

return ICounter(\_counter).count();

}

}

// Uniswap example

interface UniswapV2Factory {

function getPair(

address tokenA,

address tokenB

) external view returns (address pair);

}

interface UniswapV2Pair {

function getReserves()

external

view

returns (uint112 reserve0, uint112 reserve1, uint32 blockTimestampLast);

}

contract UniswapExample {

address private factory = 0x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f;

address private dai = 0x6B175474E89094C44Da98b954EedeAC495271d0F;

address private weth = 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2;

function getTokenReserves() external view returns (uint, uint) {

address pair = UniswapV2Factory(factory).getPair(dai, weth);

(uint reserve0, uint reserve1, ) = UniswapV2Pair(pair).getReserves();

return (reserve0, reserve1);

}

}