**Inheritance in Solidity**

**Introduction:**

In this lesson, we are going to introduce the concept of inheritance and overriding, two powerful tools that allow developers to create more modular, maintainable, and reusable smart contracts. By leveraging these techniques, you can build upon existing contracts and customize their functions.

**Inheritance:**

We are going to enhance the `SimpleStorage` contract by adding a new functionality: the ability to add five (5) to the stored `favoriteNumber`.

To achieve this, we could duplicate the existing `SimpleStorage` contract and make changes to the new version. However, this approach leads to redundant code. A better practice could be to utilize inheritance, which is the mechanism that allows the `AddFiveStorage` contract to derive all the functionalities of `SimpleStorage`.

Let’s begin by creating a new file `AddFiveStorage.sol` and name-importing `SimpleStorage.sol`:

SPDX-License-Identifier: MIT

Pragma solidity ^0.8.18;

Import {SimpleStorage} from “./SimpleStorage.sol”;

Contract AddFiveStorage is SimpleStorage {}

The `is` keyword signifies inheritance and links the parent contract `SimpleStorage` to its child contract, `AddFiveStorage`.

**Override and virtual:**

The `AddFiveStorage` contract now inherits all methods from `SimpleStorage`. It’s possible to add new functions to it, for example:

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

Return “Hello”;

}

We can also modify existing functions from `SimpleStorage` by using the `override`keyword. Let’s say that we want to modify the `store` function, adding ‘5’ to the favorite number being stored. If we copy the exact signature of the `store` function, an error will occur:

function store(uint256 \_newFavNumber) public {}

TypeError: Overriding function is missing “override” specifier.

**Note:** To override a method from the parent contract, we must replicate the exact function signature, including its name, parameters and adding the visibility and the `override` keyword to it:

function store(uint256 \_newFavNumber) public override {}

Yet, another error will pop up:

TypeError: Trying to override a non-virtual function.

To address this, we need to mark the `store` function in `SimpleStorage.sol` as ‘virtual’, enabling it to be overridden by child contracts:

function store(uint256 favNumber) public virtual {

// function body

}

Finally, we can add the new functionality to the `store` function in `AddFiveStorage`, allowing it to add ‘5’ to the stored `favoriteNumber`:

function store(uint256 \_newFavNumber) public override {

favoriteNumber = \_newFavNumber + 5;

}

**Conclusion:**

In this lesson, we utilized inheritance to modify the `SimpleStorage` contract, without rewriting all its code. After deploying the contract `AddFiveStorage` and storing the number 2, it will return the `favoriteNumber` 7. This confirms that the `store` function in `AddFiveStorage` contract successfully overrides the existent `store` function in `SimpleStorage`.