**Interacting with Contract ABI**

**Introduction:**

In this lesson, the `StorageFactory` contract will be upgraded to keep track of all deployed `SimpleStorage` contracts. This will allow us also to interact with each deployed contract individually.

**Storing the deployed contracts:**

In the current `StorageFactory` version, every time `createSimpleStorageContract` is called, a new `SimpleStorage` contract is deployed and ‘overridden’ inside the variable `SimpleStorage`. Past deployments are not being tracked down.

To solve this issue we can create a variable `listOfSimpleStorageContracts`, which is an array of `SimpleStorage` contracts. In this way, whenever a contract is created, it gets added to a dynamic array.

SimpleStorage[] public listOfSimpleStorageContracts;

We can then modify the function `createSimpleStorageContract`, pushing the newly deployed contract to this variable.

function createSimpleStorageContract() public {

SimpleStorage simpleStorageContractVariable = new SimpleStorage();

listOfSimpleStorageContracts.push(simpleStorageContractVariable);

}

In Remix, you can access `listOfSimpleStorageContracts` via the index of type `uint256`, which refers to the position of the deployed contract inside the dynamic array.

**Simple Storage interaction:**

`StorageFactory` can interact with the deployed contracts by calling their `store` function. To do this we need to create a function `sfStore`:

function sfStore(uint256 \_simpleStorageIndex, uint256 \_simpleStorageNumber) public {

//SimpleStorage store function will be called here

}

**Important:** Every time you have to interact with another contract, you need:

1. The contract address

2. The contract \*\*ABI (Application Binary Interface)\*\*: a standardized way for interacting with the binary version of a smart contract deployed on the blockchain. It specifies the functions, their parameters, and the structure of the data that can be used to interact with the contract. It’s generated by the compiler.

**Note:** If you do not have the full ABI available, a function selector will suffice (see later in the course).

If you go to Solidity’s compile tab, you will find a button that lets you copy the ABI to the clipboard.

**Note:** In Solidity, it’s possible to type cast an address to a type contract

We can now proceed to store a new number on a `SimpleStorage` contract:

function sfStore(uint256 \_simpleStorageIndex, uint256 \_simpleStorageNumber) public {

listOfSimpleStorageContracts[\_simpleStorageIndex].store( \_simpleStorageNumber);

}

We can then retrieve the stored value with a get function:

function sfGet(uint256 \_simpleStorageIndex) public view returns (uint256) {

// return SimpleStorage(address(simpleStorageArray[\_simpleStorageIndex])).retrieve();

Return listOfSimpleStorageContracts[\_simpleStorageIndex].retrieve();

}

**Conclusion:**

The `StorageFactory` contract was able to create a list of `SimpleStorage` contracts, store a variable in each of them and read it back.