**>SimpleStorage.sol**

// SPDX-License-Identifier: MIT

pragma solidity ^0.6.0;

contract SimpleStorage {

// uint256 favoriteNumber = 5;

// bool favoriteBool = false;

// string favoriteString = "String";

// int256 favoriteInt = -5;

// bytes32 favoriteBytes = "cat";

uint256 public favoriteNumber;

struct People{

uint favoriteNumber;

string name;

}

People public person = People({favoriteNumber : 23, name : "Khushbu"});

function store(uint256 \_favoriteNumber) public{

favoriteNumber = \_favoriteNumber;

}

function retrieve() public view returns(uint256)

{

return favoriteNumber;

}

People[] public people;

mapping(string=>uint256)public nameToFavoriteNumber;

function addperson(string memory \_name , uint256 \_favoriteNumber) public {

people.push(People(\_favoriteNumber, \_name));

nameToFavoriteNumber[\_name]=\_favoriteNumber;

}

}

**>StorageFactory.sol**

// SPDX-License-Identifier: MIT

pragma solidity ^0.6.0;

import "./SimpleStorage.sol";

contract StorageFactory is SimpleStorage

{

SimpleStorage[] public simpleStorageArray;

function createSimpleStorageContract()public

{

SimpleStorage simplestorage=new SimpleStorage();

simpleStorageArray.push(simplestorage);

}

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

//Address

//ABI

SimpleStorage simpleStorage=SimpleStorage(address(simpleStorageArray[\_simpleStorageIndex]));

simpleStorage.store(\_simpleStorageNumber);

}

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

{

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

// SimpleStorage simpleStorage=SimpleStorage(address(simpleStorageArray[\_simpleStorageIndex]));

//return simpleStorage.retrieve();

}

}

**>FundMe.sol**

// SPDX-License-Identifier: MIT

pragma solidity >=0.6.6 <0.9.0;

import "@chainlink/contracts/src/v0.6/interfaces/AggregatorV3Interface.sol";

import "@chainlink/contracts/src/v0.6/vendor/SafeMathChainlink.sol";

contract FundMe

{

using SafeMathChainlink for uint256;

mapping(address => uint256) public adderssToAmountFunded;

address[] public funders;

address public owner;

constructor() public

{

owner = msg.sender;

}

function fund()public payable

{

uint256 minimumUSD=50\*10\*18;

require(getConversionRate(msg.value)>=minimumUSD,"You need to spend more ETH!");

adderssToAmountFunded[msg.sender] +=msg.value;

//What the ETH -> USD conersion rate

funders.push(msg.sender);

}

function getVersion() public view returns (uint256){

AggregatorV3Interface priceFeed = AggregatorV3Interface(0x8A753747A1Fa494EC906cE90E9f37563A8AF630e);

return priceFeed.version();

}

function getPrice() public view returns(uint256){

AggregatorV3Interface priceFeed = AggregatorV3Interface(0x8A753747A1Fa494EC906cE90E9f37563A8AF630e);

(,int256 answer,,, ) = priceFeed.latestRoundData();

// uint80 roundID,

// int256 price,

// uint256 startedAt,

// uint256 timeStamp,

// uint80 answeredInRound)

// = priceFeed.latestRoundData();

return uint256(answer \* 10000000000);

}

function getConversionRate(uint256 ethAmount) public view returns(uint256)

{

uint256 ethPrice = getPrice();

uint256 ethAmountInUsd = (ethPrice \* ethAmount) / 100000000000000;

return ethAmountInUsd;

}

modifier onlyOwner

{

require(msg.sender == owner);

\_;

}

function withdraw() payable onlyOwner public

{

// require(msg.sender == owner);

msg.sender.transfer(address(this).balance);

for (uint256 funderIndex=0; funderIndex < funders.length; funderIndex++)

{

address funder = funders[funderIndex];

adderssToAmountFunded[funder] = 0;

}

funders = new address[](0);

}

}