Introduction to DeFi

09 / 13

Agenda

Solidity & Foundry

0 Speaker Introduction

1 Learning Resource

2 Environment Configuration

3 Introduction to Foundry

4 Smart Contract Breakdown

What is smart contract and why we choose Solidity?

There are many more smart contract language we can choose from, including Vyper, Rust, and more

Learning Resource

How to start learning Solidity in 2024?

Basics

- Cyfrin Updraft
- Crypto Zombie
- WTF-Academy
- Solidity by Example
- Solidity Official Documentation

Advance

- Decentralized Finance @UC Berkely
- Secureum
- Calyptus
- Finematics

Environment Configuration

How can you develop smart contract?

Remix Online IDE

- Gas Estimation Issue
- Hard for collaboration

Hardhat - javaScript

- Good for frontend integration.
- Extensive plugin ecosystem via npm

★ Foundry - Rust based

- Blazing fast testing framework
- Solidity-like syntax

We recommend using macOS or Linux.
If you are using Windows, consider installing WSL2.

Prerequisites

- Git / GitHub Account
- Rust
- Foundryup

Teaching Materials

All the course materials are available in the GitHub organization.

Foundry Tools

Forge

For testing purpose

Cast

interacting with EVM smart contracts

Anvil

Local Ethereum node

Project Layout

/src

All the smart contracts are located here.

/test

Place all the unit tests and integration tests.

/script

Put your deployment scripts in this folder

Demo Time

Project Learning

We will learn by reading a simple smart contract codebase.

Simple Bank Contract

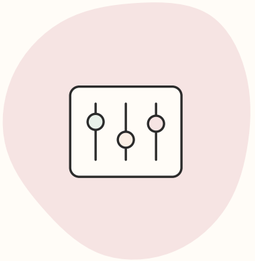
Please check this repo

Functionality

A simple contract that stores your ETH token

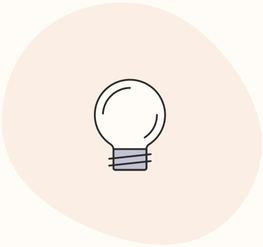
Clone Project	git clone <SSH-URL>
Build the Project	forge install && forge build
Run tests	forge test -vvv

Simple Bank



Deposit

Store the Ether into the Bank Contract

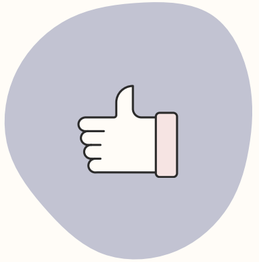


Withdraw

Take your deposited ether out

RUGPULL

Withdraw all ethers locked in the contract



Anything Else?

What function do you want to add

Codebase Review

Please spend some time reading the entire codebase

Smart Contract Breakdown

License

- A comment for specifying the License

Version

- What is the current Solidity Version

Interface

- Difference between the function and interface?

Contract

- How many components are inside a contract

SPDX-License-Identifier

This should be included in all .sol files; otherwise, you will encounter a compiler error.

Details can be found in the SPDX-License documentation.

Version

There are many Solidity compiler versions, and we should specify which one we want to use.

1. There is slight difference among distinct versions

   0.4.22 -> constructor, 0.8.0 -> safeMath

2. Include the pragma version in every file.

   Locking the version is preferable, except for libraries.

3. Pattern: pragma solidity x.y.z

   pragma solidity ^0.8.3 : [0.8.3, 0.9.0)

   pragma solidity >=0.8.3 <0.8.7

4. Please use the latest Solidity version 0.8.26

Interface

define a set of function declarations without implementing their functionality

1. Compare the functions in the contract and the interface.

2. There are several restrictions in interface:
   - all declared functions must be external
   - cannot declare a constructor, receive, fallback
   - cannot declare state variables

3. Interface tell us how to interact with other contract.

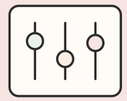
Simple Bank Address: 0x5C69...5aA6f

✓ IUnsafeBank(address).deposit()

✗ address.deposit

State Variable

You can find them at the top of the contract.



Data Type

uint256, address,
mapping, string

Visibility

public, internal,
private

Mutability

immutable,
constant, payable

Data Type

Solidity is statically-type language, which means the type of each variable needs to be specified in code at compile time.

Data Type (1) - Value Type

*There are two types of accounts: EOA & CA

Type	Example	Byte	Default Value
Boolean	True / False	1 Byte	False
Unsigned Integer	uint128, uint256	uint256 - 32 bytes	0
Integer	int128, int256	int256 - 32 bytes	0
address*	0x5C69...5aA6	20 bytes	address(0)
Fixed-Size bytes array	bytes1, bytes2, ..., bytes32	bytes32 - 32 bytes	bytes32(0)

EOA

Externally Owned Account

For example, Wallet Address

CA

Contract Account

For example, Simple Bank Contract

Data Type (2) - Reference Type

Type	Example
Array	uint256[], string, bytes (Dynamic Size Bytes Array)
Mapping	mapping(address=>uint256), mapping(address addr=>uint), mapping(address addr=>uint balance)
Struct	struct Demo {uint256 x, uint256 y}

Visibility

In Solidity, you can control access to functions and state variables in contract, as well as how to interact with them.

Visibility

1 public

- Compiler automatically generates getter function, it allows external and internal access.

2 internal

- Can only be access within the contract they are defined and the derived contracts.
- Default visibility for state variables.
- They can not access externally.

3 private

- Resemble the internal visibility but cannot be accessed in derived contract.

Private Visibility

NOTE!!!
Private Visibility State Variable Can Still Be Retrieved Through Off-Chain Mechanism.

1. Check out this contract, the state variables are marked private and internal
2. Calculate the storage slots of these state variables to determine where the variables are stored.
3. Use Alchemy Composer to look up their values.

Mutability

State mutability refers to a function's capability to modify the state of a contract.

Mutability

1 constant

- The variables cannot be modified after the contract has been deployed.
- The value has to be fixed at compile-time.
- Check `VERSION` variable in simple bank.

2 immutable

- The variables cannot be modified after the contract has been deployed.
- It can be assigned during construction.
- Check `_owner` variable in simple bank

3 payable

- Used in `address` type, specifying whether the address can receive ethers.

Rest Time

We finally finish the state variable section.

Constructor

Something similar to `def __init__()` if you are familiar with Python.

1. Please look at the constructor in the simple bank contract.
2. payable modifier enable contracts to receive ether when initialization
3. Immutable state variable should be set in the construction phase
4. Additional Note: No constructor keyword before Solidity 0.4.22 compiler version

Functions

There are visibility and mutability for functions as well.

1. Please look at the `balances` function in the simple bank contract.
2. Please look at other functions and check their visibilities
   -> public, external, internal, private
3. Please look at other functions and check their mutabilities
   -> pure, view, payable
4. Additional Note: No constructor keyword before Solidity 0.4.22 compiler version

Function Visibility

1. public: Any account can call
   -> Be careful with access control issue
2. external: Only other contracts and account can call
   -> It can be bypassed with `this.f()`
3. internal: Can be called inside contract and child contracts
4. private: Only inside the contract that defines the function

Function Mutability

1. view: this function only read but no write operations in state variable.
2. pure: this function does not read nor write state variable
3. payable: enable this function to receive ethers

Example

Case 1

ftrace | funcSig

```
function returnMultiple() public pure returns(uint256, bool, uint256[3] memory) {
    return (1, true, [uint256(1), 2, 3]);
}
```

Case 2

ftrace | funcSig

```
function returnMultiple() public pure returns(uint256 _number, bool _bool, uint256[3] memory _arr) {
    _number = 1;
    _bool = true;
    _arr = [uint256(1), 2, 3];
}
```

Functionality

Let's check each function and its implementation

Deposit

Receives Ether provided by the user and updates the balance.

1. Please look at the `deposit` function in the simple bank contract.

2. Global Variables
   - msg.sender: Account that invokes the function.
   - msg.value: The amount of ether transferred. The function receives ether should be `payable`.
   - msg.data: The raw data sent in the transaction.

3. After the operation, an event is emmited to indicate the success of deposit

Event

A way to log and notify external entities

1. Please check the `unsafeBank__depositToken` event in the simple bank contract.
2. Events are created with `emit` keyword
3. Event can be seemed as logging in EVM, it can not be accessed from within contracts
4. Subscribe and listen to these event through the RPC interface of an Ethereum client.
5. Up to three parameters can be indexed, which help filter logs by the indexed parameter.

WHAT IF?

An user mistakenly send ether to the contract
without the usage of `deposit` function

receive

When other accounts send ethers directly to contracts, it will be triggered.

1. The receive function is executed when a call to the contract is made with empty calldata for a pure ether token transfer.

2. receive is a special function and should be declared as: receive() external payable {...}

3. A contract can only have one receive function

receive

When other accounts send ethers directly to contracts, it will be triggered.

4 It can not have argument and return value.

5 The function keyword is not required, but the payable mutability and external visibility are necessary.

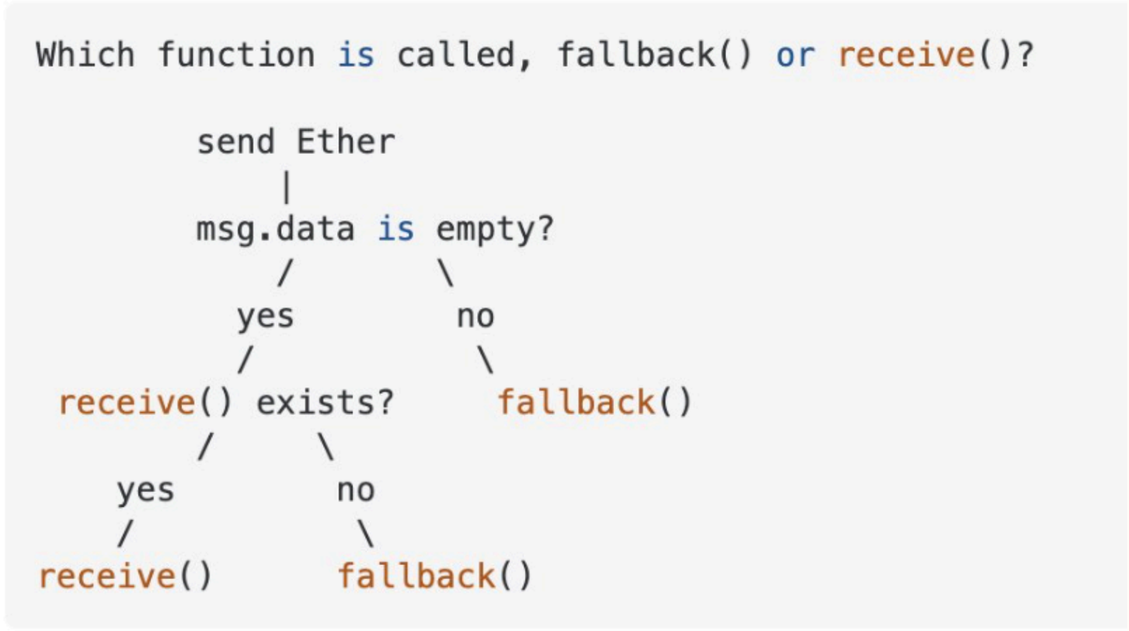
fallback

When other accounts make external call to contracts, it will be triggered.

1. The fallback function is executed on an invocation
   (1) No matched function signature.
   (2) No receive function when receiving ethers
2. A contract can only have one fallback function
3. The function keyword is not required, but the payable mutability and external visibility are necessary.

Difference Between receive and fallback

Fallback and Receive in Solidity



Withdraw

Take out the Ether stored by the user and updates the balance.

1. Please look at the `withdraw` function in the simple bank contract.

2. It first examines whether the user have sufficient token balance.
   -> If not, an error occurs

3. It then transfer the ether back to the user and check whether the procedure succeed.
   -> What is `call` and how to use it?

4. Finally, the balance is updated and an event is emitted.

Error

After Solidity 0.8.4, it provide gas-efficient way to explain errors

1. Check `unsafeBank__notEnoughBalance` error in the simple bank contract.

2. Use `revert` to trigger an error, it consume less gas than `require`. (Check L#55)

   -> After Solidity 0.8.26, they are the same

3. There are several ways to implement error.

Error

Method 1

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.4;

error Unauthorized();

contract VendingMachine {
    address payable owner = payable(msg.sender);

    function withdraw() public {
        if (msg.sender != owner)
            revert Unauthorized();

        owner.transfer(address(this).balance);
    }
    // ...
}
```

Error

Method 2

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.4;

/// Insufficient balance for transfer. Needed `required` but only
/// `available` available.
/// @param available balance available.
/// @param required requested amount to transfer.
error InsufficientBalance(uint256 available, uint256 required);

contract TestToken {
    mapping(address => uint) balance;
    function transfer(address to, uint256 amount) public {
        if (amount > balance[msg.sender])
            // Error call using named parameters. Equivalent to
            // revert InsufficientBalance(balance[msg.sender], amount);
            revert InsufficientBalance({
                available: balance[msg.sender],
                required: amount
            });
        balance[msg.sender] -= amount;
        balance[to] += amount;
    }
    // ...
}
```

withdraw

Address Type

There are several built-in methods for address type variable

1 Built-in Members

- balance: ether balance
- code: compiled byte code
- codehash: hashing result of the code stored

2 Built-in Methods

- call
- staticcall
- delegatecall

3 Built-in Methods for address payable

- call: returns success status and return data, forward almost all the gas.
- send: returns false on failure, 2300 gas limit
- transfer: reverts on failure, 2300 gas limit

Ether Transfer

There are three ways to send ethers,
but please follow best practice.

1 Recommended Method: Using `call`

(bool success, bytes memory data) =
address.call{value: amount}(“”)

2 Remember to check the boolean value

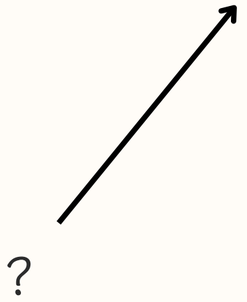
require(success, “transfer failed”);

3 Why not using `transfer` or `send`

- Insufficient Gas Supply
- Gas limit is originally used for protection

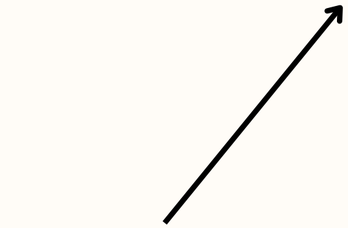
What is this?

```
(bool success, bytes memory data) = address.call{value: amount}("")
```



External Call

```
(bool success, bytes memory data) = address.call{value: amount}("")
```



msg.data

External Call

There are two ways for external call

1 IUnsafeBank(address).deposit()

We define the interface for function invoking.

2 (bool success,) = address.call(data)

We use low-level call for interaction

withdraw

Function Dispatching

Function

function withdraw(uint256 amount) external;

Function Signature

withdraw(uint256)

Function Selector

bytes4(keccak256(“withdraw(uint256)”))

msg.data

Function Selector + Calldata

msg.data

There are two ways of generating msg.data value.

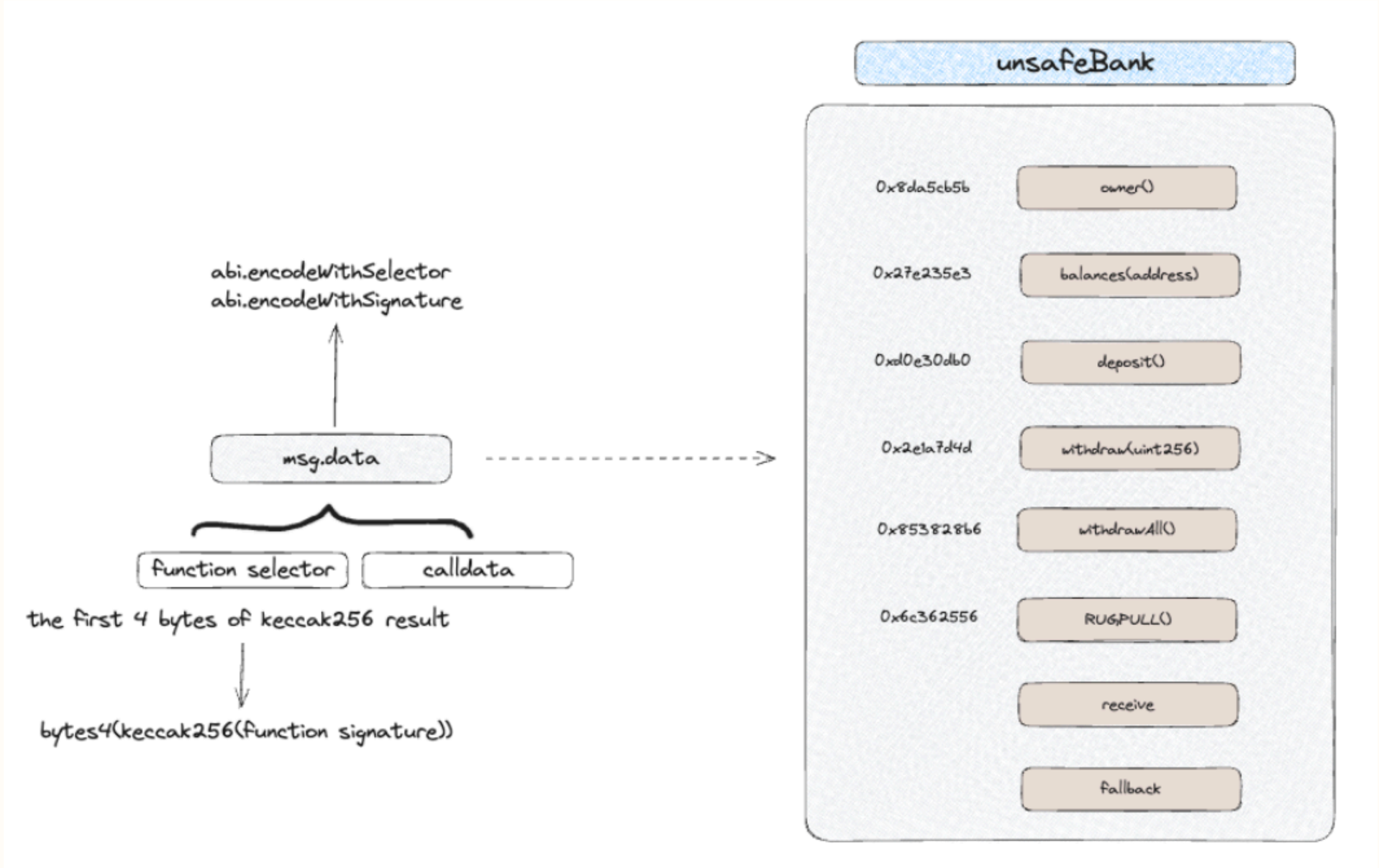
1. abi.encodeWithSignature
2. abi.encodeWithSelector

```
// External call using interface
IUnsafeBank(address(bank)).deposit{value: 1 ether}();

// External call using low-level call
bytes memory data = abi.encodeWithSignature("deposit()");
(bool success, ) = address(bank).call{value: 1 ether}(data)
require(success, "failed");
```

withdraw

Function Dispatching



RUGPULL

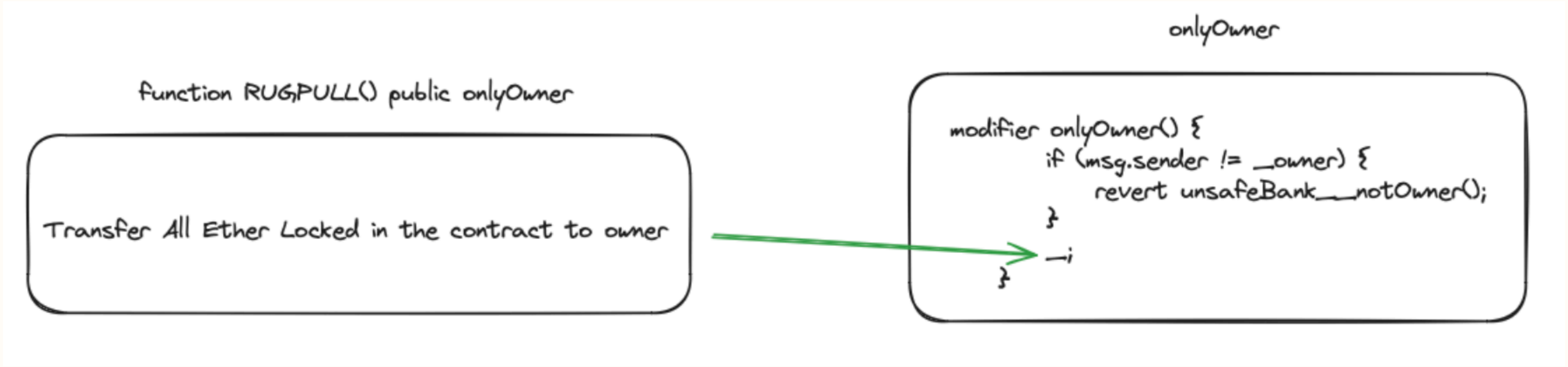
The protocol owner will takeover all the token in the contract.

1. Please look at the `RUGPULL` function in the simple bank contract.

2. There is a modifier called `onlyOwner`

   -> Please check the implementation details

Modifier


function RUGPULL() public onlyOwner
Transfer All Ether Locked in the contract to owner
onlyOwner
modifier onlyOwner() {
if (msg.sender != _owner) {
revert unsafeBank__notOwner();
}
_;
}

Foundry Test

How do we test the smart contract?

Category

There are different types of testing methods to secure your contract

1. Unit Test: tests isolated source code to validate its expected behavior.
2. Integration Test
2. Invariant Test
2. Fuzzing Test
2. Formal Verification Test

Testing Breakdown

License / Version

- The same as our source contract

Package Import

- Testing packages and source contract

Contract

- Inherit the `Test` contract, what is it?

function

- What is the purpose of `setUp` function?

Test

The helper library that includes all the things you need for testing.

1. vm: vm.prank(), vm.load(), ...
2. console: console.log(), ...
2. assertion: assertEq(), assertLt(), ...
2. deal: deal(), ...
2. Other: expectRevert(), expectEmit(), ...

Rules

What is the structure of running a foundry test through forge

1. Put all the tests in the `test/` folder
2. All the tests have a “test” prefix
3. Before each test is executed, the setUp function will run.
4. Run all the test by “forge test”
5. Check the result in the Terminal

forge test

What is the structure of running a foundry test through forge

1. -vvv: Print execution traces for failing tests
2. -vvvv: Print execution traces for all tests and setup traces for failing tests
3. -vvvvv: Print execution and setup traces for all tests

forge test

What is the structure of running a foundry test through forge

1. --match-test: Only run test functions matching the specified regex pattern.
2. --match-contract: Only run tests in contracts matching the specified regex pattern.
3. --match-path: Only run tests in source files matching the specified glob pattern.

makeAddr

We need to define some roles, such as admin, user, hacker when testing

1. Please check the `setUp` function and the address variables.
2. Creates an address derived from the provided variable.
3. You need various roles during the contract development.

Prank

How do I perform actions in the context of ADMIN?

1. Please check the `setUp` function
2. Recap: `msg.sender` is the account who invoke this function.
3. Sets `msg.sender` to the specified address for the next call.
4. Use `startPrank` and `stopPrank` for the following function calls.

Assertion

To check if the execution result matches your expectation

1. Please check `test_version` function
2. assertEq, assertTrue, assertLt, ...
3. Please check std assertions for more info

Event

We can also verify if the test is emitted as expected

1. Please check `test_deposit` function
2. Assert a specific log is emitted during the next call.
3. Some specific rules - link
4. Similar assertions: expectCall, expectRevert

Deal

What if you need to execute an operation as if you have 500 Ether?

1. Please check `test_deposit` function
2. Set the token balance of the account, supporting both ERC-20 and Ether.
3. deal(address to, uint256 give);
   deal(address token, address to, uint give)

Homework 1

Assignment Guideline

- Documentation
- You are not allowed to modify any protected files.
- You are not allowed to use any foundry cheat code in the assignment, except for the logging library.
- You are not allowed to directly change the default system configuration.

Protected Files

- We have listed all the protected file paths.
- if any of these paths are modified, you will receive zero points.

LiaoToken

- ERC-20
- Current Library: Oppenzeppelin, Solady, Solmate
- Different Version: ERC20Burnable, ERC20Permit, ERC20Upgradeable, ...
- Additional Resource: https://github.com/d-xo/weird-erc20

NFinTech

- ERC-721
- Current Library: Oppenzeppelin, Solady, Solmate
- Different Version: ERC721Enumerable, ERC721Burnable, ...
- Additional Resource: https://github.com/chiru-labs/ERC721A

NFinTech

- ERC-721
- Current Library: Oppenzeppelin, Solady, Solmate
- Different Version: ERC721Enumerable, ERC721Burnable, ...
- Additional Resource: https://github.com/chiru-labs/ERC721A