

John Miguel P. Miclat

NW-301

Week 3

Constants:

The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' sidebar is open, showing a gas limit of 3,000,000 and a value of 0 Wei. The 'CONTRACT' section displays the code for 'Constants - Solidity 2/Constants.sol'. The code defines two constants: `MY_ADDRESS` (an address) and `MY_UINT` (a uint256). The 'Deploy' button is highlighted. The 'Transactions recorded' and 'Deployed Contracts' sections show the deployment of the contract at address 0xd9f1...39138. The 'Low level interactions' section shows the creation of the contract and calls to its functions.

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.20;
contract Constants {
    address public constant MY_ADDRESS = 0x77788889999aAAbBbCcCcDddeeEffffCcC;
    uint public constant MY_UINT = 123;
}
```

This screenshot shows the same Remix IDE interface, but with a different contract named 'Var - Solidity 2/Constants.sol'. The code defines a variable `MY_ADDRESS` of type address. The 'Transactions recorded' and 'Deployed Contracts' sections show the deployment of the contract at address 0xd9f1...39138. The 'Low level interactions' section shows the creation of the contract and calls to its functions.

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.20;
/*Contract Constants */
address public constant MY_ADDRESS = 0x77788889999aAAbBbCcCcDddeeEffffCcC;
/**/
contract Var {
    address public MY_ADDRESS = 0x77788889999aAAbBbCcCcDddeeEffffCcC;
}
```

Mapping:

```

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract MappingExample {
    // Define a mapping to store values associated with addresses
    mapping(address => uint256) public valueMapping;

    // Function to set a value for the sender's address
    function setValue(uint256 _value) public {
        valueMapping[msg.sender] = _value;
    }

    // Function to retrieve the value associated with the sender's address
    function getvalue() public view returns (uint256) {
        return valueMapping[msg.sender];
    }
}

```

Deployed Contracts

- CONSTANTS AT 0x091...39138
- VAR AT 0x088...39fA8 (MEMORY)
- MAPPINGEXAMPLE AT 0x8E5...
- Balances: 0 ETH
- setValue: 100
- getvalue: 100
- 0x uint256: 100
- valueMapping: address

Low level interactions

CALLDATA

Transactions recorded

Explain contract

From: 0x883B0ea701c568545dCfc803FcB875F50bed0C4
to: MappingExample.getValue() 0x883B0ea701c568545dCfc803FcB875F50bed0C4
execution cost: 2496 gas (cost only applies when called by a contract)
input: 0x209...65255
output: 0x00
decoded input: ()
decoded output: ()
logs: []
raw logs: []

Error Handling:

```

contract ErrorHandling{
    mapping(address => uint256) public valueMapping;

    // Function to set a value for the sender's address
    function setValue(uint256 _value) public {
        // Ensure the value being set is not zero
        require(_value != 0, "Value cannot be zero");
        valueMapping[msg.sender] = _value;
    }

    // Function to retrieve the value associated with the sender's address
    function getvalue() public view returns (uint256) {
        // Ensure the sender has set a value before retrieving
        require(valueMapping[msg.sender] != 0, "No value set for sender");
        return valueMapping[msg.sender];
    }
}

```

Deployed Contracts

- CONSTANTS AT 0x091...39138
- VAR AT 0x088...39fA8 (MEMORY)
- MAPPINGEXAMPLE AT 0x8E5...
- ERRORHANDLING AT 0x8E5...
- Balances: 0 ETH
- setValue: 200
- getvalue: 200
- 0x uint256: 200
- valueMapping: address
- 0x uint256: 200

Low level interactions

CALLDATA

Transactions recorded

Explain contract

creation of ErrorHandling pending...
transaction to ErrorHandling.setvalue pending ...
call to ErrorHandling.getvalue pending ...
call to ErrorHandling.valueMapping ...
call to ErrorHandling.setvalue ...
call to ErrorHandling.getvalue ...
call to ErrorHandling.valueMapping ...

Debug sessions:

- [call] From: 0x883B0ea701c568545dCfc803FcB875F50bed0C4 to: ErrorHandling.(constructor) value: 0 wei data: 0x000...e0033 logs: 0 hash: 0xae0...2b5c4
- [tx] From: 0x883B0ea701c568545dCfc803FcB875F50bed0C4 to: ErrorHandling.setValue() value: 0 wei data: 0x532...000c8 logs: 0 hash: 0x630...6ef14
- [call] From: 0x883B0ea701c568545dCfc803FcB875F50bed0C4 to: ErrorHandling.getValue() value: 0 wei data: 0x209...65255 logs: 0 hash: 0x2e0...e0034

Ownable Demo:

The screenshot shows the Remix IDE interface. On the left, there's a sidebar with tabs for 'DEPLOY & RUN TRANSACTIONS' and 'Transactions recorded'. Under 'Transactions recorded', it says 'Run transactions using the latest compilation result' with 'Save' and 'Run' buttons. Below that is a section titled 'Deployed Contracts' with a balance of '0 ETH'. It lists three functions: 'anyOneCanCall', 'onlyOwnerCan...', and 'setOwner'. The 'setOwner' function has a value of '150'. At the bottom of the sidebar, there's a 'Deploy' button and a 'Deploy at Address' dropdown. The main area shows the Solidity code for the 'Ownable' contract. The code includes a constructor that sets the owner to the sender, a modifier 'onlyOwner' that checks if the sender is the owner, and three external functions: 'setOwner', 'onlyOwnerCanCallThisFunc', and 'anyOneCanCall'. The 'setOwner' function has a gas limit of 222200. The 'onlyOwnerCanCallThisFunc' and 'anyOneCanCall' functions have gas limits of 2595. The code is annotated with comments like 'creation of ownable pending...' and transaction logs. The bottom right corner shows the date and time as '2/11/2025 1:47 pm'.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Ownable {
    address public owner;

    constructor() {
        owner = msg.sender;
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "not owner");
    }

    function setOwner(address _newOwner) external onlyOwner {
        require(_newOwner != address(0), "invalid address");
        owner = _newOwner;
    }

    function onlyOwnerCanCallThisFunc() external onlyOwner {
        // code
    }

    function anyOneCanCall() external {
        // code
    }
}
```

REFLECTION QUESTIONS

- **Function Modifiers & Ownable**

When should you use a modifier like onlyOwner instead of inline checks, and what risks arise if ownership isn't managed properly?

- Inline checks and modifiers have a similar form of validation control or ownership, but modifiers are more flexible in terms of an open construction of data rather than inline check could help you alone for contract size for particular scenarios.

- **Error Handling**

How do you choose between require, revert, and assert, and why might custom errors be better than error strings?

- Custom errors are more helpful because they can be more helpful in giving out the proper specific information regarding to an error that may occur. Nonetheless, using the advantage of every behavior of require, revert, and assert are one of the tools that could help the user to know the very factor of an error in the code.

- **Constants & Variables**

When should a value be constant, immutable, or mutable, and how does that choice affect gas cost and flexibility?

- It is a scale system where you could choose for more flexibility but costly or a hard code that is not costly of your gas. Depending on your usage or block of code, you could use every advantage you could do in order to make your system more manageable and efficient in all its aspects. Therefore, being able to properly asses on the right approach that you could take is like a form of satisfaction for every users that you could have, in order for them to be in a comfortable approach of your system.