HW0

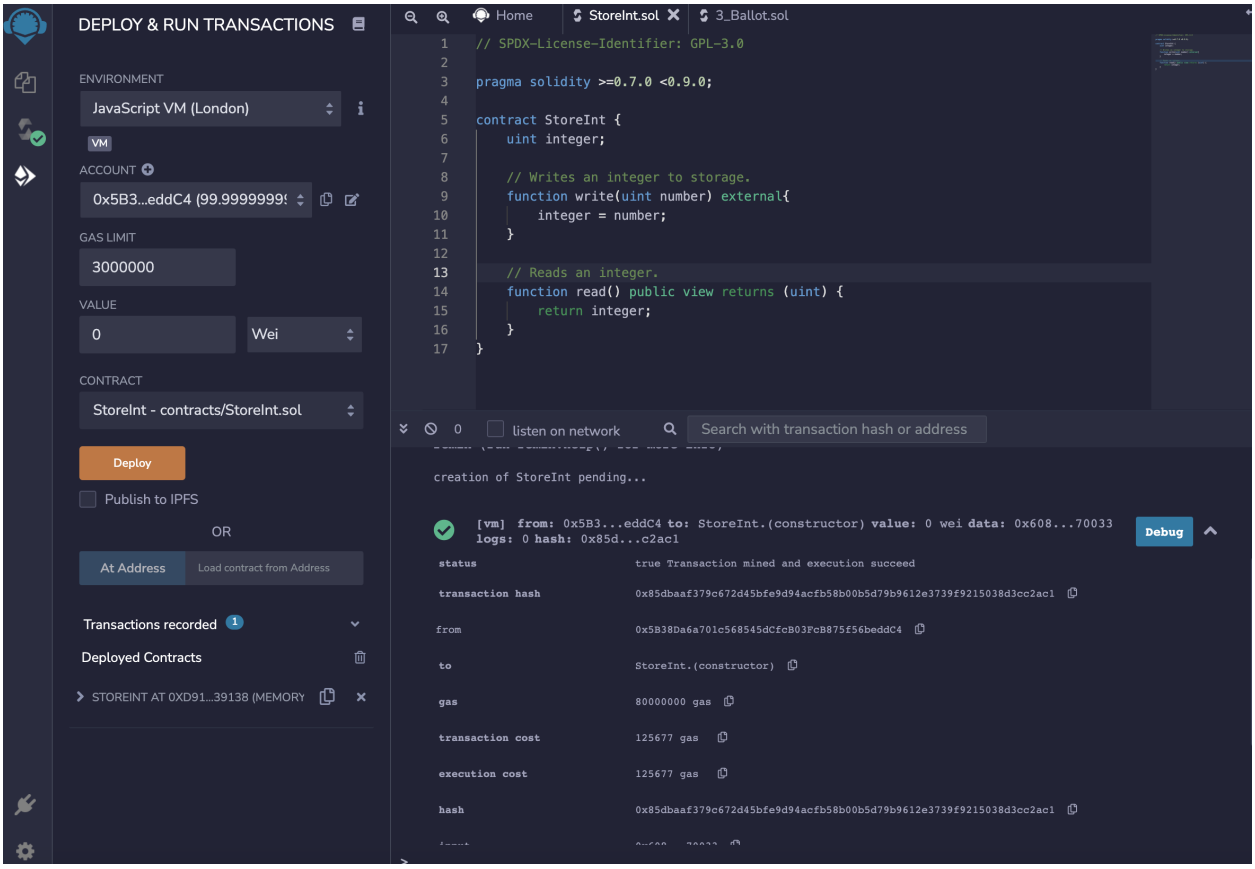
- registration email: jsrshark110@gmail.com
- discord username: Sara J#0578
- github: https://github.com/katesonia

1. Program a super simple “Hello World” smart contract: store an unsigned integer and then retrieve it. Please clearly comment on your code. Once completed, deploy the smart contract on Remix. Include the .sol file and a screenshot of the Remix UI once deployed in your final submission pdf (more info about submission formatting below)

`StoreInt.sol`

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

contract StoreInt {
    uint integer;

    // Writes an integer to storage.
    function write(uint number) external{
        integer = number;
    }

    // Reads an integer.
    function read() public view returns (uint) {
        return integer;
    }
}
```

Deployed at:



1. On the documentation page, the “Ballot” contract demonstrates a lot of features on Solidity. Read through the script and try to understand what each line of code is doing, then implement the Possible Improvements by **reducing the number of transactions in the “giveRightToVote” function while maintaining the same functionality of the program**.

1. Deploy your script on Remix and compare the difference in gas fees between the original script and the improved script when giving 10 voters the right to vote. Once completed, submit (1) your improved version of the contract as a .sol file with comments describing the changes you made, and (2) screenshots (before and after) of the gas fees for the transaction(s) to give 10 voters the right to vote.

(1) `BallotImproved.sol`

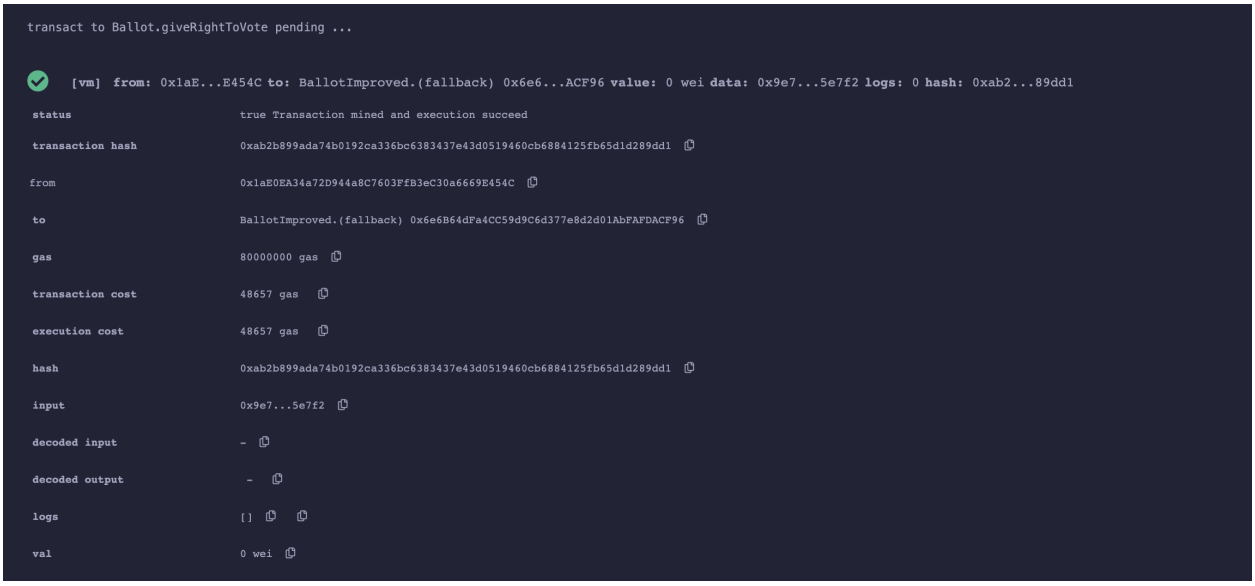
```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

/**
 * @title Ballot
 * @dev Implements voting process along with vote delegation
 */
contract BallotImproved {

    struct Voter {
        uint weight; // weight is accumulated by delegation
        bool voted;  // if true, that person already voted
```

```
        address delegate; // person delegated to
        uint vote;   // index of the voted proposal
    }

    struct Proposal {
        // If you can limit the length to a certain number of bytes,
        // always use one of bytes1 to bytes32 because they are much cheaper
        bytes32 name;   // short name (up to 32 bytes)
        uint voteCount; // number of accumulated votes
    }

    address public chairperson;

    mapping(address => Voter) public voters;

    Proposal[] public proposals;

    /**
     * @dev Create a new ballot to choose one of 'proposalNames'.
     * @param proposalNames names of proposals
     */
    constructor(bytes32[] memory proposalNames) {
        chairperson = msg.sender;
        voters[chairperson].weight = 1;

        for (uint i = 0; i < proposalNames.length; i++) {
            // 'Proposal({...})' creates a temporary
            // Proposal object and 'proposals.push(...)'
            // appends it to the end of 'proposals'.
            proposals.push(Proposal({
                name: proposalNames[i],
                voteCount: 0
            }));
        }
    }

    /**
     * @dev Give 'voter' the right to vote on this ballot. May only be called by 'chairperson'.
     * @param voterAddresses list of voter addresses
     * ASSIGNMENT: changed input parameters to be a list of addresses instead of a single address.
     */
    function giveRightToVote(address[] memory voterAddresses) public {
        require(
            msg.sender == chairperson,
            "Only chairperson can give right to vote."
        );

        for (uint i=0; i < voterAddresses.length; i++) {
            address voter = voterAddresses[i];
            require(
                !voters[voter].voted,
                "The voter already voted."
            );
            require(voters[voter].weight == 0);
            voters[voter].weight = 1;
        }
    }

    /**
     * @dev Delegate your vote to the voter 'to'.
     * @param to address to which vote is delegated
     */
    function delegate(address to) public {
        Voter storage sender = voters[msg.sender];
        require(!sender.voted, "You already voted.");
        require(to != msg.sender, "Self-delegation is disallowed.");

        while (voters[to].delegate != address(0)) {
```

```
            to = voters[to].delegate;

            // We found a loop in the delegation, not allowed.
            require(to != msg.sender, "Found loop in delegation.");
        }
        sender.voted = true;
        sender.delegate = to;
        Voter storage delegate_ = voters[to];
        if (delegate_.voted) {
            // If the delegate already voted,
            // directly add to the number of votes
            proposals[delegate_.vote].voteCount += sender.weight;
        } else {
            // If the delegate did not vote yet,
            // add to her weight.
            delegate_.weight += sender.weight;
        }
    }

    /**
     * @dev Give your vote (including votes delegated to you) to proposal 'proposals[proposal].name'.
     * @param proposal index of proposal in the proposals array
     */
    function vote(uint proposal) public {
        Voter storage sender = voters[msg.sender];
        require(sender.weight != 0, "Has no right to vote");
        require(!sender.voted, "Already voted.");
        sender.voted = true;
        sender.vote = proposal;

        // If 'proposal' is out of the range of the array,
        // this will throw automatically and revert all
        // changes.
        proposals[proposal].voteCount += sender.weight;
    }

    /**
     * @dev Computes the winning proposal taking all previous votes into account.
     * @return winningProposal_ index of winning proposal in the proposals array
     */
    function winningProposal() public view
            returns (uint winningProposal_)
    {
        uint winningVoteCount = 0;
        for (uint p = 0; p < proposals.length; p++) {
            if (proposals[p].voteCount > winningVoteCount) {
                winningVoteCount = proposals[p].voteCount;
                winningProposal_ = p;
            }
        }
    }

    /**
     * @dev Calls winningProposal() function to get the index of the winner contained in the proposals array and then
     * @return winnerName_ the name of the winner
     */
    function winnerName() public view
            returns (bytes32 winnerName_)
    {
        winnerName_ = proposals[winningProposal()].name;
    }
}
```

(2)

Before improvement, giving right to 1 voter costs 48657 gas, giving rights to 10 voters costs **486570** gas



After improvement, giving rights to 10 voters in one transaction costs **279280** gas, which is much less than before.