ASSIGNMENT: Error Handling & Restrictions

Your assignment for this lecture would be to go back to the contracts you have made by now and think about the restrictions you can make you user inputs and numbers.

Are there any functions that might overflow or underflow a number, or would you want this function to be accessible only by the owner of the contract?

Use the appropriate function to restrict that from happening and after that continue to the next lecture.

---------------------

Contracts from this lecture:

**IMPORTANT!**

**In the presentation I made a mistake "**accounts[msg.sender] += money;" **should be** "accounts[msg.sender] += msg.value;" and in the **Math**contract I forgot the specify that the **variable "c" is uint.**

The contracts below should work with solidity ^0.5.0 and with ^0.4.24. There is no change there.

Contract Require Section

1. pragma solidity ^0.5.0;
2. contract Bank {
3. mapping(address => uint) public accounts;
5. function deposit() public payable {
6. require(accounts[msg.sender] + msg.value >= accounts[msg.sender], “Overflow error”);
7. accounts[msg.sender] += msg.value;
8. }
10. function withdraw(uint money) public {
11. require(money <= accounts[msg.sender]);
12. accounts[msg.sender] -= money;
13. }
14. }

Contract Revert Section

1. pragma solidity ^0.5.0;
3. contract Bank {
4. mapping(address => uint) public accounts;
6. function deposit() public payable {
7. if(accounts[msg.sender] + msg.value <= accounts[msg.sender]) {
8. revert("Overflow error");
9. }
10. accounts[msg.sender] += msg.value;
11. }
13. function withdraw(uint money) public {
14. if(money <= accounts[msg.sender]){
15. //can have more if statements
16. revert();
17. }
18. accounts[msg.sender] -= money;
19. }
20. }

Contract Assert Section

1. pragma solidity^0.5.0;
3. contract Math {
5. function add(uint256 a, uint256 b) internal pure returns (uint) {
6. uint c = a + b;
7. assert(c >= a);
8. return c;
9. }
11. function multiply(uint256 a, uint256 b) internal pure returns (uint) {
12. if (a == 0) {
13. return 0;
14. }
15. uint c = a \* b;
16. assert(c / a == b);
17. return c;
18. }
19. }