

# Lab: Error Handling in Solidity

## Prerequisites

1. Chrome or Firefox browser.
2. An Internet connection
3. Open Remix with the following Smart Contract:

```
pragma solidity >=0.5.11 <0.7.0;

contract ExceptionExample {

    mapping(address => uint) public balanceReceived;

    function receiveMoney() public payable {
        balanceReceived[msg.sender] += msg.value;
    }

    function withdrawMoney(address payable _to, uint _amount) public {
        if(_amount <= balanceReceived[msg.sender]) {
            balanceReceived[msg.sender] -= _amount;
            _to.transfer(_amount);
        }
    }
}
```

## Step by Step Instruction

*Deploy the Smart Contract in the JavaScript VM*

Open the “Deploy and Run Transactions” view in Remix with the smart contract

*Try to withdraw more than you have*

At the beginning you have 0 Ether. When we try to withdraw some then simply nothing happens.

The screenshot displays the Remix Ethereum IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' panel is active, showing the 'JavaScript VM' environment. The account selected is '0xCA3...a733c (99.9)'. The gas limit is set to 3,000,000 and the value is 0 wei. The contract 'ExceptionExample' is deployed to the browser. Below the deployment section, the 'Deployed Contracts' list shows 'ExceptionExample at 0x692...77b3A (memory)'. The 'receiveMoney' function is highlighted in red, and a red arrow points to the 'withdrawMoney' function in the list. The 'balanceReceived' variable is also visible. On the right, the Solidity code for 'ExceptionExample.sol' is shown, featuring a mapping for 'balanceReceived', a 'receiveMoney' function, and a 'withdrawMoney' function that uses an 'if' statement to check the balance before withdrawing. The bottom panel shows the transaction log, which includes the creation of the contract and the execution of the 'withdrawMoney' function. A red circle highlights the transaction log entry for the 'withdrawMoney' function, showing the transaction hash and the state of the contract after the withdrawal.

```
1 pragma solidity ^0.5.11;
2
3 contract ExceptionExample {
4
5     mapping(address => uint) public balanceReceived;
6
7     function receiveMoney() public payable {
8         balanceReceived[msg.sender] += msg.value;
9     }
10
11
12     function withdrawMoney(address payable _to, uint _amount) public {
13         if(_amount <= balanceReceived[msg.sender]) {
14             balanceReceived[msg.sender] -= _amount;
15             _to.transfer(_amount);
16         }
17     }
18 }
19
20
```

creation of ExceptionExample pending...

[vm] from:0xca3...a733c to:ExceptionExample.(constructor) value:0 wei data:0x608...b

transact to ExceptionExample.withdrawMoney pending ...

[vm] from:0xca3...a733c to:ExceptionExample.withdrawMoney(address,uint256) 0x692... hash:0x402...95288

*Replace the if with a Require*

```

pragma solidity >=0.5.11 <0.7.0;

contract ExceptionExample {

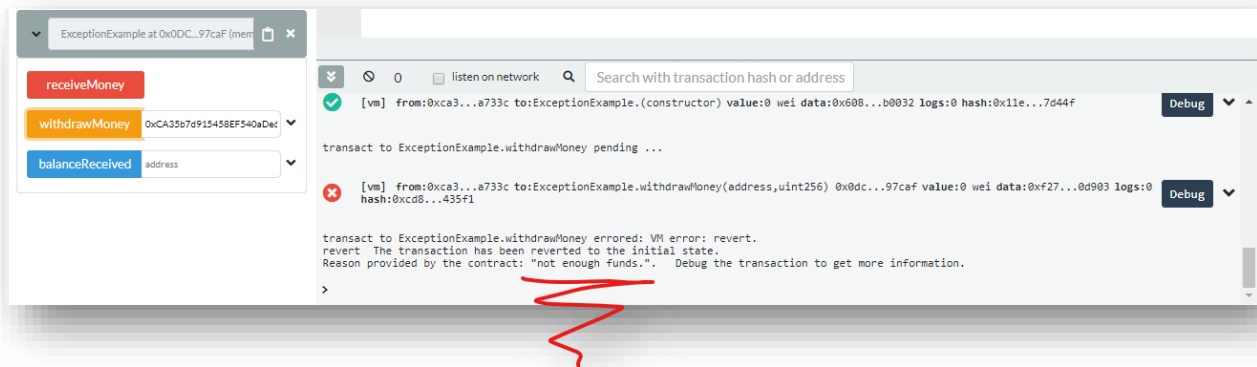
    mapping(address => uint) public balanceReceived;

    function receiveMoney() public payable {
        balanceReceived[msg.sender] += msg.value;
    }

    function withdrawMoney(address payable _to, uint _amount) public {
        require(_amount <= balanceReceived[msg.sender], "not enough funds.");
        balanceReceived[msg.sender] -= _amount;
        _to.transfer(_amount);
    }
}

```

And try to withdraw again. It should throw an error!



### Add an Assert

Imagine your balance is not of the type uint256, but of the type uint64 – to save some storage costs.

If you send two times 10 Ether to your smart contract it will automatically roll over to 0:

```

pragma solidity >=0.5.11 <0.7.0;

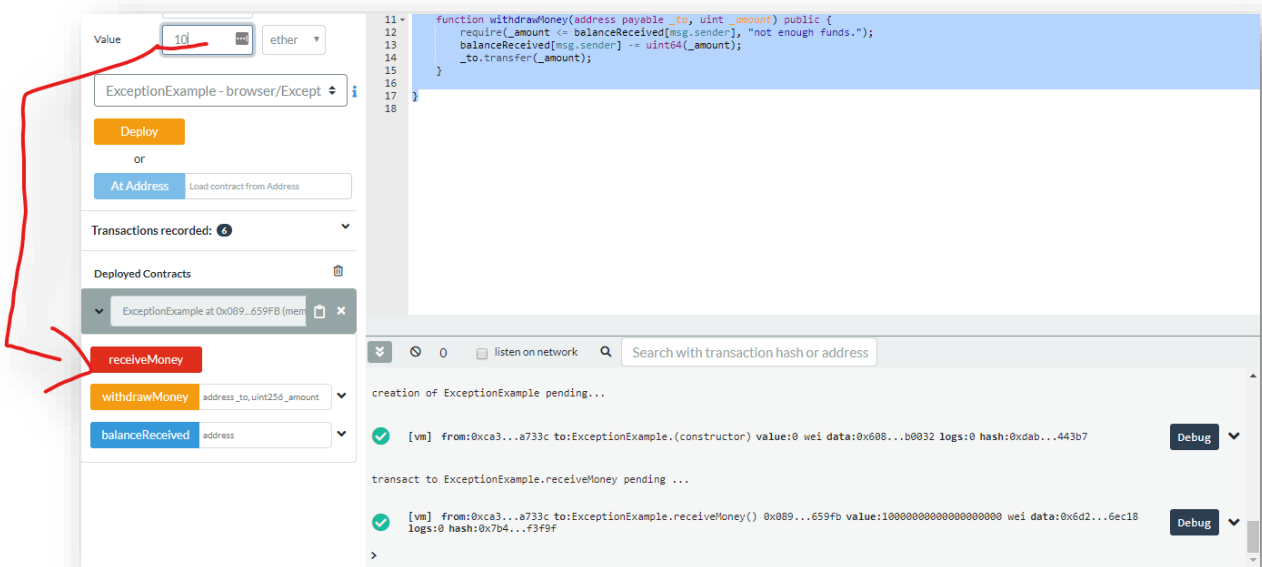
contract ExceptionExample {

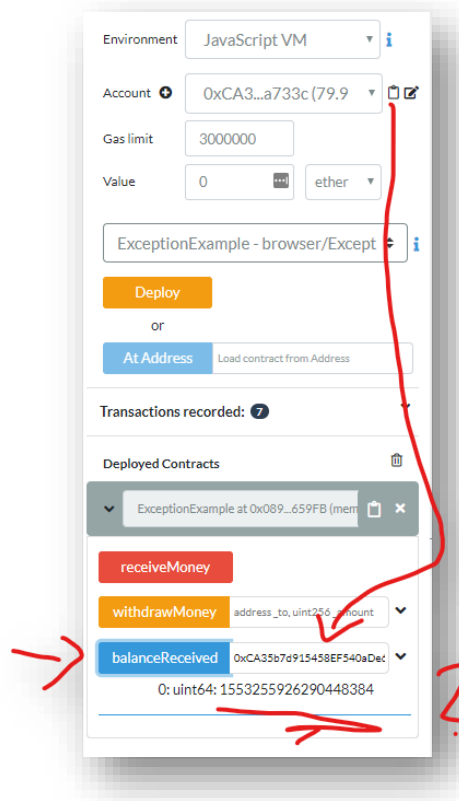
    mapping(address => uint64) public balanceReceived;

    function receiveMoney() public payable {
        balanceReceived[msg.sender] += uint64(msg.value);
    }

    function withdrawMoney(address payable _to, uint _amount) public {
        require(_amount <= balanceReceived[msg.sender], "not enough funds.");
        balanceReceived[msg.sender] -= uint64(_amount);
        _to.transfer(_amount);
    }
}

```





It is lower, because the uint64 rolls over to 0 after reaching the maximum value of 18446744073709551616 – that's around 18.44 Ether

Here we could use asserts to make sure we don't roll over – in both directions! When withdrawals happen, we don't suddenly have more balance available than before and when deposits happen that the balance after depositing is really higher than before.

*Add an assert, to make sure the balance can only grow larger*

```
pragma solidity >=0.5.11 <0.7.0;

contract ExceptionExample {

    mapping(address => uint64) public balanceReceived;

    function receiveMoney() public payable {
        assert(balanceReceived[msg.sender] + uint64(msg.value) >= balanceReceived[msg.sender]);
        balanceReceived[msg.sender] += uint64(msg.value);
    }

    function withdrawMoney(address payable _to, uint _amount) public {
        require(_amount <= balanceReceived[msg.sender], "not enough funds.");
        assert(balanceReceived[msg.sender] >= balanceReceived[msg.sender] - _amount);
        balanceReceived[msg.sender] -= uint64(_amount);
        _to.transfer(_amount);
    }
}
```

[Test again](#)

Send again 2 times 10 Ether to the smart contract. It will end in an error.

### Deployed Contracts

▼ ExceptionExample at 0x8c1...401f5 (memo) 🗑️ ✕

receiveMoney

withdrawMoney

 ▼

balanceReceived

 ▼

⌵
 0 listen on network 🔍

Search with transaction hash or address

logs:0 hash:0x487...41cf5

transact to ExceptionExample.receiveMoney pending ...

❌ [vm] from:0xca3...a733c to:ExceptionExample.receiveMoney() 0x8c1...401f5 value:10000000000  
 logs:0 hash:0x62c...a61ba

transact to ExceptionExample.receiveMoney errored: VM error: invalid opcode.  
 invalid opcode  
 The execution might have thrown.  
 Debug the transaction to get more information.

>

# **Congratulations, LAB is completed**

