👛 Solidity Project 👛
Shared Wallet with Allowance Function

Real-World Use-Case for this Project

- Allowance for Children per day/week/month to be able to spend a certain amount of funds.
- Employers give employees an allowance for their travel expenses.
- Businesses give contractors an allowance to spend a certain budget.

Development-Goal

- Have an on-chain wallet smart contract.
- This wallet contract can store funds and let users withdraw again.
- You can also give "allowance" to other, specific user-addresses.
- Restrict the functions to specific user-roles (owner, user)
- Re-Use existing smart contracts which are already audited to the greatest extent

Contents

Step 1 – We Define the Basic Smart Contract

This is the very basic smart contract. It can receive Ether and it's possible to withdraw Ether, but all in all, not very useful quite yet. Let's see if we can improve this a bit in the next step.

```
pragma solidity ^0.5.13;

contract SharedWallet {

    function withdrawMoney(address payable _to, uint _amount) public {
        _to.transfer(_amount);
    }

    function() external payable {

    }
}
```

Step 2 – Permissions: Allow only the Owner to Withdraw Ether

In this step we restrict withdrawal to the owner of the wallet. How can we determine the owner? It's the user who deployed the smart contract.

```
pragma solidity ^0.5.13;

contract SharedWallet {

    address owner;

    constructor() public {
        owner = msg.sender;
    }

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

    function withdrawMoney(address payable _to, uint _amount) public onlyOwner {
        _to.transfer(_amount);
    }

    function() external payable {

    }
}
```

Step 3 – Permissions: Use Re-Usable Smart Contracts from OpenZeppelin

Having the owner-logic directly in one smart contract isn't very easy to audit. Let's break it down into smaller parts and re-use existing audited smart contracts from OpenZeppelin for that.

```
pragma solidity ^0.5.13;

import "https://github.com/OpenZeppelin/openzeppelin-
contracts/contracts/ownership/Ownable.sol";

contract SharedWallet is Ownable {

    function withdrawMoney(address payable _to, uint _amount) public onlyOwner {
        _to.transfer(_amount);
    }

    function() external payable {

    }
}
```

Step 4 – Permissions: Add Allowances for External Roles

In this step we are adding a mapping so we can store address => uint amounts. This will be like an array that stores [0x123546…] an address, to a specific number. So, we always know how much someone can withdraw. We also add a new modifier that checks: Is it the owner itself or just someone with allowance?

```
pragma solidity ^0.5.13;

import "https://github.com/OpenZeppelin/openzeppelin-
contracts/contracts/ownership/Ownable.sol";

contract SharedWallet is Ownable {

    mapping(address => uint) public allowance;

    function addAllowance(address _who, uint _amount) public onlyOwner {
        allowance[_who] = _amount;
    }

    modifier ownerOrAllowed(uint _amount) {
        require(isOwner() || allowance[msg.sender] >= _amount, "You are not allowed!");
        _;
    }

    function withdrawMoney(address payable _to, uint _amount) public ownerOrAllowed(_amoun
t) {
        require(_amount <= address(this).balance, "Contract doesn't own enough money");
        _to.transfer(_amount);
    }

    function() external payable {

    }
}
```

Step 5 – Improve/Fix Allowance to avoid Double-Spending

Without reducing the allowance on withdrawal, someone can continuously withdraw the same amount over and over again. We have to reduce the allowance for everyone other than the owner.

```
function reduceAllowance(address _who, uint _amount) internal ownerOrAllowed(_amount) {
        allowance[_who] -= _amount;
    }

    function withdrawMoney(address payable _to, uint _amount) public ownerOrAllowed(_amoun
t) {
        require(_amount <= address(this).balance, "Contract doesn't own enough money");
        if(!isOwner()) {
            reduceAllowance(msg.sender, _amount);
        }
        _to.transfer(_amount);
    }
```

Step 6 – Improve Smart Contract Structure

Now we know our basic functionality, we can structure the smart contract differently. To make it easier to read, we can break the functionality down into two distinct smart contracts.

```
pragma solidity ^0.5.13;

import "https://github.com/OpenZeppelin/openzeppelin-
contracts/contracts/ownership/Ownable.sol";

contract Allowance is Ownable {

    mapping(address => uint) public allowance;

    function setAllowance(address _who, uint _amount) public onlyOwner {
        allowance[_who] = _amount;
    }

    modifier ownerOrAllowed(uint _amount) {
        require(isOwner() || allowance[msg.sender] >= _amount, "You are not allowed!");
        _;
    }

    function reduceAllowance(address _who, uint _amount) internal ownerOrAllowed(_amount)
{
        allowance[_who] -= _amount;
    }

}

contract SharedWallet is Allowance {

    function withdrawMoney(address payable _to, uint _amount) public ownerOrAllowed(_amoun
t) {
        require(_amount <= address(this).balance, "Contract doesn't own enough money");
        if(!isOwner()) {
            reduceAllowance(msg.sender, _amount);
        }
        _to.transfer(_amount);
    }

    function() external payable {

    }
}
```

Step 7 – Add Events in the Allowances Smart Contract

One thing that's missing is events.

```
pragma solidity ^0.5.13;

import "https://github.com/OpenZeppelin/openzeppelin-
contracts/contracts/ownership/Ownable.sol";

contract Allowance is Ownable {

    event AllowanceChanged(address indexed _forWho, address indexed _byWhom, uint _oldAmou
nt, uint _newAmount);
    mapping(address => uint) public allowance;

    function setAllowance(address _who, uint _amount) public onlyOwner {
        emit AllowanceChanged(_who, msg.sender, allowance[_who], _amount);
        allowance[_who] = _amount;
    }

    modifier ownerOrAllowed(uint _amount) {
        require(isOwner() || allowance[msg.sender] >= _amount, "You are not allowed!");
        _;
    }

    function reduceAllowance(address _who, uint _amount) internal ownerOrAllowed(_amount)
{
        emit AllowanceChanged(_who, msg.sender, allowance[_who], allowance[_who] - _amount
);
        allowance[_who] -= _amount;
    }

}

contract SharedWallet is Allowance {
    //…
}
```

Step 8 – Add Events in the SharedWallet Smart Contract

```
contract SharedWallet is Allowance {

    event MoneySent(address indexed _beneficiary, uint _amount);
    event MoneyReceived(address indexed _from, uint _amount);

    function withdrawMoney(address payable _to, uint _amount) public ownerOrAllowed(_amoun
t) {
        require(_amount <= address(this).balance, "Contract doesn't own enough money");
        if(!isOwner()) {
            reduceAllowance(msg.sender, _amount);
        }
        emit MoneySent(_to, _amount);
        _to.transfer(_amount);

    }

    function() external payable {
        emit MoneyReceived(msg.sender, msg.value);
    }
}
```

Step 9 – Add the SafeMath Library safeguard Mathematical Operations

```
pragma solidity ^0.5.13;

import "https://github.com/OpenZeppelin/openzeppelin-
contracts/contracts/ownership/Ownable.sol";
import "https://github.com/OpenZeppelin/openzeppelin-
contracts/contracts/math/SafeMath.sol";

contract Allowance is Ownable {

    using SafeMath for uint;

    event AllowanceChanged(address indexed _forWho, address indexed _byWhom, uint _oldAmou
nt, uint _newAmount);
    mapping(address => uint) public allowance;

    function setAllowance(address _who, uint _amount) public onlyOwner {
        //...
    }

    modifier ownerOrAllowed(uint _amount) {
        //...
    }

    function reduceAllowance(address _who, uint _amount) internal ownerOrAllowed(_amount)
{
        emit AllowanceChanged(_who, msg.sender, allowance[_who], allowance[_who].sub(_amou
nt));
        allowance[_who] = allowance[_who].sub(_amount);
    }

}

contract SharedWallet is Allowance {
    //...
}
```

Step 10 – Remove the Renounce Ownership functionality

```
contract SharedWallet is Allowance {

    //...

    function renounceOwnership() public onlyOwner {
        revert("can't renounceOwnership here"); //not possible with this smart contract
    }

    //...
}
```

Step 11 – Move the Smart Contracts into separate Files

Allowance.sol:

```
pragma solidity ^0.5.13;

import "https://github.com/OpenZeppelin/openzeppelin-
contracts/contracts/ownership/Ownable.sol";
import "https://github.com/OpenZeppelin/openzeppelin-
contracts/contracts/math/SafeMath.sol";

contract Allowance is Ownable {
     //...
}
```

SharedWallet.sol:

```
pragma solidity ^0.5.13;

import "https://github.com/OpenZeppelin/openzeppelin-
contracts/contracts/ownership/Ownable.sol";
import "https://github.com/OpenZeppelin/openzeppelin-
contracts/contracts/math/SafeMath.sol";
import "./Allowance.sol";

contract SharedWallet is Allowance {
     //...
}
```

The Final Smart Contract(s)

File: Allowance.sol

```
pragma solidity ^0.5.13;

import "https://github.com/OpenZeppelin/openzeppelin-
contracts/contracts/ownership/Ownable.sol";
import "https://github.com/OpenZeppelin/openzeppelin-contracts/contracts/math/SafeMath.sol";

contract Allowance is Ownable {

    using SafeMath for uint;

    event AllowanceChanged(address indexed _forWho, address indexed _byWhom, uint _oldAmount
, uint _newAmount);
    mapping(address => uint) public allowance;

    function setAllowance(address _who, uint _amount) public onlyOwner {
        emit AllowanceChanged(_who, msg.sender, allowance[_who], _amount);
        allowance[_who] = _amount;
    }

    modifier ownerOrAllowed(uint _amount) {
        require(isOwner() || allowance[msg.sender] >= _amount, "You are not allowed!");
        _;
    }

    function reduceAllowance(address _who, uint _amount) internal ownerOrAllowed(_amount) {
        emit AllowanceChanged(_who, msg.sender, allowance[_who], allowance[_who].sub(_amount
));
        allowance[_who] = allowance[_who].sub(_amount);
    }

}
    function renounceOwnership() public onlyOwner {
        revert("can't renounceOwnership here"); //not possible with this smart contract
    }

    function() external payable {
        emit MoneyReceived(msg.sender, msg.value);
    }
}
```

File: SmartContract.sol

```
pragma solidity ^0.5.13;

import "https://github.com/OpenZeppelin/openzeppelin-
contracts/contracts/ownership/Ownable.sol";
import "https://github.com/OpenZeppelin/openzeppelin-
contracts/contracts/math/SafeMath.sol";
import "./Allowance.sol";

contract SharedWallet is Allowance {

    event MoneySent(address indexed _beneficiary, uint _amount);
    event MoneyReceived(address indexed _from, uint _amount);

    function withdrawMoney(address payable _to, uint _amount) public ownerOrAllowed(_amoun
t) {
        require(_amount <= address(this).balance, "Contract doesn't own enough money");
        if(!isOwner()) {
            reduceAllowance(msg.sender, _amount);
        }
        emit MoneySent(_to, _amount);
        _to.transfer(_amount);
    }

    function renounceOwnership() public onlyOwner {
        revert("can't renounceOwnership here"); //not possible with this smart contract
    }

    function() external payable {
        emit MoneyReceived(msg.sender, msg.value);
    }
}
```

Congratulations, LAB is completed

From the Course “Ethereum Blockchain Developer – Build Projects in Solidity”

FULL COURSE:

https://www.udemy.com/course/blockchain-developer/?referralCode=E8611DF99D7E491DFD96