SharedWallet.sol

```
//SPDX-License-Identifier: MIT

pragma solidity 0.8.1;

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

    }
}
```

Pembaruan Soliditas

Sebelum Solidity 0.6 fungsi fallback hanya disebut "function() hutang eksternal" - Fungsi tanpa nama. Sejak Solidity 0.6 ada dua fungsi yang berbeda: satu disebut fallback dan yang lainnya disebut "menerima". Hanya "menerima" yang bisa menerima eter. Anda dapat membaca lebih lanjut tentang ini di panduan saya! Juga, kode di lab ini telah di-porting ke Solidity 0.8. Perubahan yang paling menonjol adalah penghapusan pustaka SafeMath, karena Solidity 0.8 tidak melakukan rollover integer otomatis lagi. Baca lebih lanjut tentang ini di topik tentang Overflow dan Underflow. Di lab ada catatan di mana Solidity 0.87 berubah masuk.

Permissions : Hanya Mengizinkan Pemilik untuk Withdraw Ether (harus memiliki public address)

Pada langkah ini kami membatasi penarikan kepada pemilik dompet. Bagaimana kita bisa menentukan pemiliknya? Itu adalah pengguna yang menyebarkan smart contract.

```
//SPDX-License-Identifier: MIT

pragma solidity 0.8.1;

contract SharedWallet {
    address owner;

    constructor() {
        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);
    }

    receive() external payable {

    }
}
```

Lihat !! kita menambahkan modifier “onlyOwner” ke fungsi withdrawMoney.

Gunakan Smart Contract yang dapat digunakan kembali dari OpenZeppelin

Memiliki owner logic secara langsung dalam satu Smart Contract tidak mudah untuk diaudit. Mari uraikan menjadi bagian yang lebih kecil dan gunakan kembali Smart Contract yang sudah diaudit dari OpenZeppelin. OpenZeppelin contract tidak memiliki fungsi isOwner() lagi, jadi kita haru membuatnya sendiri. Perhatikan bahwa owner() adalah fungsi dari kontrak Ownable.sol

```
//SPDX-License-Identifier: MIT

pragma solidity 0.8.1;

import "https://github.com/OpenZeppelin/openzeppelin-
contracts/blob/master/contracts/access/Ownable.sol";

contract SharedWallet is Ownable {
    address owner;

    function isOwner() internal view returns(bool) {
        return owner() == msg.sender;
    }

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

    receive() external payable {

    }
}
```

Permissions : Menambahkan Tunjangan untuk Peran Eksternal

Pada langkah ini kita menambahkan pemetaan sehingga kita dapat menyimpan alamat => jumlah uint. Ini akan seperti array yang menyimpan [0x123546...] alamat, ke nomor tertentu. Jadi, kita selalu tahu berapa banyak yang bisa ditarik seseorang. Kami juga menambahkan yang baru modifikator yang memeriksa: Apakah pemiliknya sendiri atau hanya seseorang dengan uang saku?

```
//SPDX-License-Identifier: MIT

pragma solidity 0.8.1;

import "https://github.com/OpenZeppelin/openzeppelin-
contracts/blob/master/contracts/access/Ownable.sol";

contract SharedWallet is Ownable {
    function isOwner() internal view returns(bool) {
        return owner() == msg.sender;
    }

    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(_amount) {
        require(_amount <= address(this).balance, "Contract doesn't own enough
money");
        _to.transfer(_amount);
    }

    receive() external payable {

    }
}
```

Apakah Anda menangkap bug?

Lihat fungsi withdrawMoney dan pikirkan baik-baik!

Dalam kuliah berikutnya kita akan sedikit meningkatkan kontrak pintar kita dan menghindari pengeluaran ganda.

Improve/Fix Allowance to avoid Double-Spending

Tanpa mengurangi tunjangan penarikan, seseorang dapat terus menerus menarik jumlah yang sama berulang-ulang. Kami harus mengurangi tunjangan untuk semua orang selain pemilik.

```
//SPDX-License-Identifier: MIT

pragma solidity 0.8.1;

import "https://github.com/OpenZeppelin/openzeppelin-
contracts/blob/master/contracts/access/Ownable.sol";

contract SharedWallet is Ownable {
    function isOwner() internal view returns(bool) {
        return owner() == msg.sender;
    }

    mapping(address => uint) public allowance;

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

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

    receive() external payable {

    }
}
```

Improve Smart Contract Structure

Sekarang kami mengetahui fungsi dasar kami, kami dapat menyusun kontrak pintar secara berbeda. Untuk membuatnya lebih mudah dibaca, kita bisa istirahat fungsionalitas menjadi dua kontrak pintar yang berbeda.

Note

Note that since Allowance is Ownable, and the SharedWallet is Allowance, therefore by commutative property, SharedWallet is also Ownable.

```
//SPDX-License-Identifier: MIT

pragma solidity 0.8.1;

import "https://github.com/OpenZeppelin/openzeppelin-
contracts/blob/master/contracts/access/Ownable.sol";

contract Allowance is Ownable {
    function isOwner() internal view returns(bool) {
        return owner() == msg.sender;
    }

    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(_amount) {
        require(_amount <= address(this).balance, "Contract doesn't own enough
money");
        if(!isOwner()) {
            reduceAllowance(msg.sender, _amount);
        }
        _to.transfer(_amount);
    }
    receive() external payable {

    }
}
```

Kedua kontrak masih dalam file yang sama, jadi kami tidak memiliki impor (belum). Itu sesuatu untuk kuliah lain nanti. Saat ini, bagian penting untuk dipahami adalah warisan.

Add Events in the Allowances Smart Contract

```
//SPDX-License-Identifier: MIT
pragma solidity 0.8.1;

import "https://github.com/OpenZeppelin/openzeppelin-
contracts/blob/master/contracts/access/Ownable.sol";

contract Allowance is Ownable {
    event AllowanceChanged(address indexed _forWho, address indexed _byWhom,
uint _oldAmount, uint _newAmount);
    mapping(address => uint) public allowance;

    function isOwner() internal view returns(bool) {
        return owner() == msg.sender;
    }

    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 {
 //…
}
```

Add Events in the SharedWallet Smart Contract

```
//SPDX-License-Identifier: MIT
pragma solidity 0.8.1;

import "https://github.com/OpenZeppelin/openzeppelin-
contracts/blob/master/contracts/access/Ownable.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(_amount) {
        require(_amount <= address(this).balance, "Contract doesn't own enough
money");
        if(!isOwner()) {
            reduceAllowance(msg.sender, _amount);
        }
        emit MoneySent(_to, _amount);
        _to.transfer(_amount);

    }

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

Add the SafeMath Library safeguard Mathematical Operations

Directly from the SafeMath Library source code:

Operasi aritmatika dalam Solidity wrap pada overflow. Ini dapat dengan mudah menghasilkan bug, karena programmer biasanya berasumsi bahwa overflow menimbulkan kesalahan, yang merupakan perilaku standar dalam bahasa pemrograman tingkat tinggi. *SafeMath* memulihkan intuisi ini dengan mengembalikan transaksi saat operasi meluap

```
//SPDX-License-Identifier: MIT
pragma solidity 0.8.1;

import "https://github.com/OpenZeppelin/openzeppelin-
contracts/blob/master/contracts/access/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 isOwner() internal view returns(bool) {
        return owner() == msg.sender;
    }
```

```
    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(_amount));
        allowance[_who] = allowance[_who].sub(_amount);
    }
}

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

Remove the Renounce Ownership functionality

Sekarang, mari kita hapus fungsi untuk menghapus pemilik. Kami hanya menghentikan ini dengan pengembalian. Tambahkan fungsi berikut ke SharedWallet :

```
//SPDX-License-Identifier: MIT
pragma solidity 0.8.1;

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

contract SharedWallet is Allowance {
    //...

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

    //...
}
```

Move the Smart Contracts into separate Files

Sebagai langkah terakhir, mari pindahkan Smart Contract ke dalam file terpisah dan gunakan fungsi impor

```
//SPDX-License-Identifier: MIT
pragma solidity 0.8.1;

import "https://github.com/OpenZeppelin/openzeppelin-
contracts/blob/master/contracts/access/Ownable.sol";

contract Allowance is Ownable {
    event AllowanceChanged(address indexed _forWho, address indexed _byWhom,
uint _oldAmount, uint _newAmount);
    mapping(address => uint) public allowance;

    function isOwner() internal view returns(bool) {
        return owner() == msg.sender;
    }

    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;
    }
}

//SPDX-License-Identifier: MIT

pragma solidity 0.8.1;

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

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