Lecture 17 - Solidity Language

Class

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.22 <0.9.0;

contract PayFor {
```

or with inheritance

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.22 <0.9.0;

import "./Ownable.sol";

contract PayFor is Ownable {
```

For passing parameters to constructor:

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.0 <0.8.0;

contract Parent {
    public string aName;
    private uint256 aNumber;

    constructor(uint256 _importantNumber, string _name) public {
                aNumber = _imporantNumber;
        aName = _name;
    }
}
```

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.0 <0.8.0;

contract ParentTwo {
    private uint256 aNumber;

    constructor(uint256 _importantNumber) public {
                aNumber = _imporantNumber;
    }
}
```

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.0 <0.8.0;

import "./Parent.sol";
import "./ParentTwo.sol";

contract Child is Parent, ParentTwo {

    constructor(uint256 valToParent) Parent(valToParent,"constantToParent"), ParentTwo(valToParent) public {
         // Child construction code goes here
    }
}
```

With the corresponding tests code (in JavaScript)

```
...
        beforeEach(async () => {
                child = await Child.new(1234);
        });
...
```

Let's take a look at Ownable:

```
 1: pragma solidity >=0.5.2;
 2: // pragma solidity ^0.5.2;
 3:
 4: /**
 5:  * @title Ownable
 6:  * @dev The Ownable contract has an owner address, and provides basic authorization control
 7:  * functions, this simplifies the implementation of "user permissions".
 8:  */
 9: contract Ownable {
10:     address private _owner;
11:
12:     event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);
13:
14:     /**
15:      * @dev The Ownable constructor sets the original `owner` of the contract to the sender
16:      * account.
17:      */
18:     constructor () internal {
19:         _owner = msg.sender;
20:         emit OwnershipTransferred(address(0), _owner);
21:     }
22:
23:     /**
24:      * @return the address of the owner.
25:      */
26:     function owner() public view returns (address) {
27:         return _owner;
28:     }
29:
30:     /**
31:      * @dev Throws if called by any account other than the owner.
32:      */
33:     modifier onlyOwner() {
34:         require(isOwner());
35:         _;
36:     }
```

```
37:
38:     /**
39:      * @return true if `msg.sender` is the owner of the contract.
40:      */
41:     function isOwner() public view returns (bool) {
42:         return msg.sender == _owner;
43:     }
44:
45:     /**
46:      * @dev Allows the current owner to relinquish control of the contract.
47:      * It will not be possible to call the functions with the `onlyOwner`
48:      * modifier anymore.
49:      * @notice Renouncing ownership will leave the contract without an owner,
50:      * thereby removing any functionality that is only available to the owner.
51:      */
52:     function renounceOwnership() public onlyOwner {
53:         emit OwnershipTransferred(_owner, address(0));
54:         _owner = address(0);
55:     }
56:
57:     /**
58:      * @dev Allows the current owner to transfer control of the contract to a newOwner.
59:      * @param newOwner The address to transfer ownership to.
60:      */
61:     function transferOwnership(address newOwner) public onlyOwner {
62:         _transferOwnership(newOwner);
63:     }
64:
65:     /**
66:      * @dev Transfers control of the contract to a newOwner.
67:      * @param newOwner The address to transfer ownership to.
68:      */
69:     function _transferOwnership(address newOwner) internal {
70:         require(newOwner != address(0));
71:         emit OwnershipTransferred(_owner, newOwner);
72:         _owner = newOwner;
73:     }

74: }
```

And now how it is used:

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.22 <0.9.0;

// import "./Ownable.sol";
import "zeppelin-solidity/contracts/ownership/Ownable.sol";


contract PayFor is Ownable {

    struct productPriceStruct {
        uint256 price;
        bool isValue;
    }
    struct paymentsStruct {
        address listOfPayedBy;
        uint256 listOfPayments;
        uint256 payFor;
    }

    event ReceivedFunds(address sender, uint256 value, uint256 application, uint256 loc);
    event Withdrawn(address to, uint256 amount);
    event SetProductPrice ( uint256 product, uint256 minPrice );
    event LogDepositReceived(address sender);

    paymentsStruct[] private paymentsFor;
    mapping (uint256 => productPriceStruct) internal productMinPrice;
    uint256[] private listOfSKU;
    uint public balance;

    constructor() Ownable() public {
    }

    /**
     * @dev set the minimum price for a product.  Emit SetProductPrice when a price is set.
     */
    function setProductPrice(uint256 SKU, uint256 minPrice) public onlyOwner {
        productMinPrice[SKU] = productPriceStruct ( minPrice, true );
        listOfSKU.push(SKU);
        emit SetProductPrice ( SKU, minPrice );
    }

    /**
     * @return true for funds received.  Emit a ReceivedFunds event.
     */
    function receiveFunds(uint256 forProduct) public payable returns(bool) {
        // Check that product is valid
        require(productMinPrice[forProduct].isValue, 'Invalid product');
        // Validate that the sender has payed for the prouct.
        require(productMinPrice[forProduct].price <= msg.value, 'Insufficient funds for product');

        balance += msg.value;
        uint256 pos;
        pos = paymentsFor.length;
        paymentsFor.push ( paymentsStruct ( msg.sender, msg.value, forProduct ) );
        emit ReceivedFunds(msg.sender, msg.value, forProduct, pos);
        return true;
    }

    /**
     * @return the number of paymetns.
     */
    function getNPayments() public onlyOwner view returns(uint256) {
        return ( paymentsFor.length );
```

```
64:     }
65:
66:     /**
67:      * @return the address that payeed with the payment amount and what was payed for.
68:      */
69:     function getPaymentInfo(uint256 n) public onlyOwner view returns(address, uint256, uint256) {
70:         require(n >= 0 && n < paymentsFor.length, 'Invalid entry');
71:         return ( paymentsFor[n].listOfPayedBy, paymentsFor[n].listOfPayments, paymentsFor[n].payFor );
72:     }
73:
74:     /**
75:      * @return the number of Products (SKUs).
76:      */
77:     function getNSKU() public view returns(uint256) {
78:         return ( listOfSKU.length );
79:     }
80:
81:     /**
82:      * @return the price for the nth SKU and its product number.
83:      */
84:     function getSKUInfo(uint256 n) public view returns(uint256, uint256) {
85:         require(n >= 0 && n < listOfSKU.length, 'Invalid entry');
86:         uint256 sku = listOfSKU[n];
87:         return ( sku, productMinPrice[sku].price );
88:     }
89:
90:     /**
91:      * @dev widthdraw funds form the contract.
92:      */
93:     function withdraw( uint256 amount ) public onlyOwner returns(bool) {
94:         // require(address(this).balance >= amount, "Insufficient Balance for withdrawl");
95:         require(balance >= amount, "Insufficient Balance for withdrawl");
96:         // address to0 = address ( Ownable.owner() );
97:         address to0 = address ( Ownable.owner );
98:         address payable to = address ( uint160(to0) );
99:         address(to).transfer(amount);
100:         emit Withdrawn(to, amount);
101:         return true;
102:     }
103:
104:     /**
105:      * @return the amount of funds that can be withdrawn.
106:      */
107:     function getBalanceContract() public view onlyOwner returns(uint256){
108:         // return address(this).balance;
109:         return balance;
110:     }
111:
112:     /**
113:      * @return Catch and save funds for abstrc transfer.
114:      */
115:     function() external payable {
116:         require(msg.data.length == 0);
117:         emit LogDepositReceived(msg.sender);
118:     }
119:
120: }
```

```
Result {
  '0': '0x861B18623d1585eeb1aE94D0852313c366E992e8',
  '1': BN {
    negative: 0,
    words: [ 4, <1 empty item> ],
    length: 1,
    red: null
  },
  '2': BN {
    negative: 0,
    words: [ 10, <1 empty item> ],
    length: 1,
    red: null
  }
}
```

Events during this test (remember that they are cumulative)

```
    Events emitted during test:
    ---------------------------

    PayFor.OwnershipTransferred(
      previousOwner: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
      newOwner: <indexed> 0xbA241C04969585671D0dc927f354eB2938511b6f (type: address)
    )

    PayFor.SetProductPrice(
      product: 8 (type: uint256),
      minPrice: 2 (type: uint256)
    )

    PayFor.SetProductPrice(
      product: 10 (type: uint256),
      minPrice: 4 (type: uint256)
    )

    PayFor.ReceivedFunds(
      sender: 0x861B18623d1585eeb1aE94D0852313c366E992e8 (type: address),
      value: 4 (type: uint256),
      application: 10 (type: uint256),

      loc: 0 (type: uint256)
    )
```

Back to code

```
    function setProductPrice(uint256 SKU, uint256 minPrice) public onlyOwner {
```

To install the OpenZepplin stuff.

```
$  npm install --save-exact zeppelin-solidity
```

And to import it into the code:

```
import "zeppelin-solidity/contracts/ownership/Ownable.sol";
```