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 {
```

```
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:

```
 1: // SPDX-License-Identifier: MIT
 2: pragma solidity >=0.4.22 <0.9.0;
 3:
 4: import "./Ownable.sol";
 5:
 6: contract PayFor is Ownable {
 7:
 8:     struct productPriceStruct {
 9:         uint256 price;
10:         bool isValue;
11:     }
12:     struct paymentsStruct {
13:         address listOfPayedBy;
14:         uint256 listOfPayments;
15:         uint256 payFor;
16:     }
17:
18:     event ReceivedFunds(address sender, uint256 value, uint256 application, uint256 loc);
19:     event Withdrawn(address to, uint256 amount);
20:     event SetProductPrice ( uint256 product, uint256 minPrice );
21:     event LogDepositReceived(address sender);
22:
23:     paymentsStruct[] private paymentsFor;
24:     mapping (uint256 => productPriceStruct) internal productMinPrice;
25:     uint256[] private listOfSKU;
26:     uint public balance;
27:
28:     constructor() Ownable() public {
29:     }
30:
```

```
31:     /**
32:      * @dev set the minimum price for a product.  Emit SetProductPrice when a price is set.
33:      */
34:     function setProductPrice(uint256 SKU, uint256 minPrice) public onlyOwner {
35:         productMinPrice[SKU] = productPriceStruct ( minPrice, true );
36:         listOfSKU.push(SKU);
37:         emit SetProductPrice ( SKU, minPrice );
38:     }
39:
40:     /**
41:      * @return true for funds received.  Emit a ReceivedFunds event.
42:      */
43:     function receiveFunds(uint256 forProduct) public payable returns(bool) {
44:         // Check that product is valid
45:         require(productMinPrice[forProduct].isValue, 'Invalid product');
46:         // Validate that the sender has payed for the prouct.
47:         require(productMinPrice[forProduct].price <= msg.value, 'Insufficient funds for product');
48:
49:         balance += msg.value;
50:         uint256 pos;
51:         pos = paymentsFor.length;
52:         paymentsFor.push ( paymentsStruct ( msg.sender, msg.value, forProduct ) );
53:         emit ReceivedFunds(msg.sender, msg.value, forProduct, pos);
54:         return true;
55:     }
56:
57:     /**
58:      * @return the number of paymetns.
59:      */
60:     function getNPayments() public onlyOwner view returns(uint256) {
61:         return ( paymentsFor.length );

62:     }
63:
64:     /**
65:      * @return the address that payeed with the payment amount and what was payed for.
66:      */
67:     function getPaymentInfo(uint256 n) public onlyOwner view returns(address, uint256, uint256) {
68:         require(n >= 0 && n < paymentsFor.length, 'Invalid entry');
69:         return ( paymentsFor[n].listOfPayedBy, paymentsFor[n].listOfPayments, paymentsFor[n].payFor );
70:     }
71:
72:     /**
73:      * @return the number of Products (SKUs).
74:      */
75:     function getNSKU() public view returns(uint256) {
76:         return ( listOfSKU.length );
77:     }
78:
79:     /**
80:      * @return the price for the nth SKU and its product number.
81:      */
82:     function getSKUInfo(uint256 n) public view returns(uint256, uint256) {
83:         require(n >= 0 && n < listOfSKU.length, 'Invalid entry');
84:         uint256 sku = listOfSKU[n];
85:         return ( sku, productMinPrice[sku].price );
86:     }
87:
88:     /**
89:      * @dev widthdraw funds form the contract.
90:      */
91:     function withdraw( uint256 amount ) public onlyOwner returns(bool) {
92:         // require(address(this).balance >= amount, "Insufficient Balance for withdrawl");
93:         require(balance >= amount, "Insufficient Balance for withdrawl");
94:         address to0 = address ( Ownable.owner() );
95:         address payable to = address ( uint160(to0) );
96:         address(to).transfer(amount);
```

```
 97:         emit Withdrawn(to, amount);
 98:         return true;
 99:     }
100:
101:     /**
102:      * @return the amount of funds that can be withdrawn.
103:      */
104:     function getBalanceContract() public view onlyOwner returns(uint256){
105:         // return address(this).balance;
106:         return balance;
107:     }
108:
109:     /**
110:      * @return Catch and save funds for abstrc transfer.
111:      */
112:     function() external payable {
113:         require(msg.data.length == 0);
114:         emit LogDepositReceived(msg.sender);
115:     }
116:
117: }
```