Pakistan Blockchain Institute

PAKISTAN BLOCKCHAIN INSTITUTE

MODULE-2
BLOCKCHAIN AND SMART CONTRACT BASICS

Class-6

Raja Rizwan Saleem
Lead Blockchain Trainer

Edversity.

Structure of a contract

A contract consists of the following multiple constructs

1. State variables
2. Function definitions
3. **Enumeration definitions**
4. Structure definitions
5. Modifier definitions
6. Event declarations

3. Enumeration

Enumeration

1. Enumeration is a custom user defined data type
2. The enum keyword is used to declare enumerations.
3. Enumerations help in declaring a custom user-defined data type in Solidity.
4. enum consists of an enumeration list, a predetermined set of named constants.
5. Constant values within an enum can be explicitly converted into integers in Solidity.

Enumeration

5. Each constant value gets an integer value, with the first one having a value of 0 and the value of each successive item is increased by 1.
6. Enums cannot be declared within functions but can be used in function

Enumeration Declaration

```
enum gender {
male,  female

}
```

1. There are two constant male and female
2. male will have value 0
3. female will have value 1

Enumeration variable declaration and initialization

```
enum gender {
    male,
    female

}
gender studentGender = gender.male;
```

Enumeration in function returns

```
enum gender {
    male,
    female

}
function getValues() public pure returns(gender){
    return gender.female;

}
```

Enumeration in function returns - Practice

```
pragma solidity ^0.8.1;

contract First{

    enum Gender {
        male,
        female
    }

    function doSomeWork() public view returns (Gender){

        Gender g = Gender.male;
        return g;
    }
}
```

Enumeration in function returns - Practice

```
pragma solidity ^0.8.1;
contract First{
    enum Gender {
        male,
        female
    }

    function doSomeWork(Gender g) public view returns (Gender){
        Gender studentGender = g;
        return studentGender;
    }
}
```

Enumeration in function returns - Practice

```
pragma solidity ^0.8.1;
contract First{
    enum Gender {
        male,
        female
    }

    function doSomeWork(Gender g) public view returns (Gender){
        Gender studentGender = g;
        return studentGender;
    }
}
```

Enumeration in function returns - Practice

```
pragma solidity ^0.5.0;
contract Test {
enum user {allowed,not_allowed,wait}
     user public u1= user.wait;
     uint public lottery = 1000;
     function owner() public {
          if (u1==user.allowed)
          {
               lottery = 0;
          }}
              function changeOwner() public {
               u1= user.not_allowed;
          }
     }
```

Enumeration in function returns - Practice

```
pragma solidity ^0.8.10;
contract Enum {
    enum Status {pending,Shipped,Accepted,Rejected,Canceled}
    Status public status;
    function get() public view returns (Status) {
        return status;
    }
    // Update status by passing uint into input
    function set(Status _status) public {
        status = _status;
    }
    // You can update to a specific enum like this
    function cancel() public {
        status = Status.Canceled;
    }
    // delete resets the enum to its first value, 0
    function reset() public {
        delete status;
    }
}
```

4.Structure/Structs

Structure / Structs

1. Structures or structs helps implement custom user-defined data types.
2. A structure is a composite data type, consisting of multiple variables of different data types.
3. Very similar to contracts; however, they do not contain any code within them.
4. Consist of only variables.

Structure / Structs

5. If you want to store information about an employee, say the employee name, age, marriage status, and bank account numbers.
6. To represent this, these individual variables related to singleemployee, a structure in Solidity can be declared using the struct keyword.
7. ***new*** keyword is not required to create instance of struct

Structure Declaration

```
struct Employee {
    string name; uint
    age;
    bool isMarried;
}
```

1. Variables inside struct can only be declared
2. This will create custom data type named Employee which is composed of multiple built-in datatype

Structure variable declaration and initialization

```
struct Employee {
    string name;
    uint age;

    bool isMarried;
}

Employee   emp1 =   Employee("Shafique",24, false);
Employee   emp2 =   Employee("Yasir",20,  true);
Employee emp3  = Employee("Alishba",26, false);
```

Structure / Structs - rules / keypoints

1. Struct cannot be passed in function argument
2. Struct cannot be returns from function
3. Struct can have can contains array, enum and mapping variables
4. mapping and arrays can also store values of type struct.

Structure

```
struct Employee {
string name;
uint age;

    bool isMarried;
    uint[] bankAccountsNumbers;
}
```

Note: we cant make functions in Structs

Struct in Practice

```
pragma solidity ^0.8.1;
contract First{
struct Student {
    string name;
    uint age;
    bool isFeepaid;
}
Student stu = Student ("Rizwan", 24,
function manageStruct() public view returns (uint) {
    int a = 56;
    Student memory s1 = Student("Rehan",20, true);
    Student memory s2 = Student("Ahmed",10, false);
    return s2.age;
}}
```

Struct in Practice

```
pragma solidity ^0.8.1;
contract First{

    struct Student {
        string name ;
        uint age ;
        bool isFeePaid;
    }

Student stu = Student ("Rizwan", 45, false);
function manageStruct() public view returns (uint){

    Student memory     s1 = Student ("Rana Sb", 40, true);
    Student memory s2 = Student ("Rizwan", 20, true);
    Student memory s3 = Student ("Shahid", 80, false);
    return s2.age;
}
}
```

Struct in Practice

```
pragma solidity ^0.8.1; contract First{
struct Student {
    string name;
    uint
    age;
    bool isFeepaid;
}
Student stu = Student ("Rizwan", 24,
function manageStruct() public view returns (uint) {
    int a = 56;
    Student memory s2 = Student("Abdullah",10, false);
    return s2.age;
}
function manageStruct2() public view returns (uint) {
    s2.name = "Absar ul Haq";
    s2.age = 38;
    return s2.age;
}}
```

Struct in Practice

```
pragma solidity >0.5.0 <0.8.21;

contract Test {
    struct Book {
        string title;
        string author;
        uint book_id;
    }
    Book book;
    function setBook() public {
        book = Book('Solidity', 'Abdullah', 10000);
    }
    function getBook() public view returns (string memory, string memory, uint) {
        return (book.title, book.author, book.book_id);
    }
}
```

Structure of a contract

A contract consists of the following multiple constructs

1. State variables
2. Function definitions
3. Enumeration definitions
4. Structure definitions
5. **Modifier definitions**
6. Event declarations

5. Modifiers

Modifiers

1. In Solidity, modifier is always associated with a function.
2. A modifier in programming languages refers to a constructthat changes the behavior of the executing code.
3. Since a modifier is associated with a function in Solidity, it has the power to change the behavior of functions that it is associated with
4. Think of modifier as a function that will be executed before execution of the target function

Modifiers

1. If we want to apply validation before executing logic of function so we can use modifier and decide in modifier whether target function should be executed.
2. This helps in writing cleaner functions without cluttering them with validation and verification rules.
3. Moreover, the modifier can be associated with multiple functions. This ensures cleaner, more readable, and more maintainable code.

Modifier Declaration

```
address owner;
modifier onlyBy(){

    if(msg.sender == owner){
        _;
    }}
```

An _ underscore in a modifier means: execute the target function. You can think of this as the underscore being replaced by the target function inline

Modifier

```
modifier verify(){
    if(age > 45){// age is state variable
        _;
    }
}
function getValues() public view verify returns(uint){
    return 56;
}
```

Modifier

```
modifier verify(uint age){
    if(age > 45){

        _;
    }}
function getValues(uint a) public view verify(a)
returns (uint) {
    return 56;
}
```

Modifier in Practice

```
pragma solidity ^0.8.1;
contract First{
    uint age = 56;
    modifier verifyAge(uint a) {
        if (a>60){
            _;
        }
    }
    function updateAge() public view verifyAge(67) returns(uint) {
    return 45;
 }
    function updateAge2() public view verifyAge(34) returns (bool) {
    return true;
    }
}
```

Structure of a contract

A contract consists of the following multiple constructs

1. State variables
2. Function definitions
3. Enumeration definitions
4. Structure definitions
5. Modifier definitions
6. **Event declarations**

6. Events

Events

1. Events are fired from contracts such that anybody interested in them can trap/catch them and execute code in response.
2. Events in Solidity are used primarily for informing the calling application about the current state of the contract by means of the logging facility of EVM.
3. They are used to notify applications about changes in contracts and applications can use them to execute their dependent logic

Events

1. Event is an inheritable member of a contract.
2. An event is emitted, it stores the arguments passed in transaction logs.
3. These logs are stored on blockchain and are accessible using address of the contract till the contract is present on the blockchain.
4. An event generated is not accessible from within contracts, not even the one which have created and emitted them.

Event Declaration and Usage

```
event ageRead(uint);

function getValues(uint a) public returns (uint){
    emit ageRead(a);

    return 56;
}
```

Event Declaration and Usage

```
pragma solidity ^0.8.1;

// Creating a contract
contract eventExample {

    // Declaring state variables
    uint256 public value = 0;

    // Declaring an event
    event Increment(address owner);

    // Defining a function for logging event
    function getValue(uint _a, uint _b) public {
        emit Increment(msg.sender);
        value = _a + _b;
    }
}
```

Event Declaration and Usage

```
contract First{

    uint age;
    event ageRead(uint);

    function updateAge(uint _age) public {

        age = _age;
        emit ageRead(age);
    }
}
```

Thanks

End of Lecture-2 (Module-6)