ETHEREUM 2.0 MASTERY PROGRAM

Instructor: Raja Rizwan Saleem

MODULE-2

BLOCKCHAIN AND SMART CONTRACT BASICS

Raja Rizwan Saleem
Lead Blockchain Trainer

Class-12

Mappings continued...
Practical Examples

Mappings Practical Examples

1. Basic Mapping Example
2. Mapping with Custom Struct
3. Nested Mapping Example
4. Mapping to store multiTypes
5. Mapping with Enum
6. Iterable Mapping (Using an Array for Keys)

Implicit / Explicit Type conversion

Type conversion

1. Solidity is a statically typed language, where variables are defined with specific data types at compile time.
2. The data type cannot be changed for the lifetime of the variable.
3. It means it can only store values that are legal for a data type.
4. For example, uint8 can store values from 0 to 255. It cannot store negative values or values greater than 255.

Type conversion

1. However, there are times when these conversions are required to copy a value into a variable of one type to another, and these are called type conversions.
2. In Solidity, we can perform Implicit and Explicit conversion

Implicit conversion

Implicit conversion means converting smaller data type into larger data type.

Conversion will be automatic and there is no need for operator or no external help is required

Implicit conversion [smaller to larger]

1. Implicit conversion are perfectly legal and there is no loss of data or mismatch of values.
2. They are completely type-safe.

Implicit conversion - Example

```
function conversion() public {
    uint8 a = 10; // Declare and initialize a uint8 variable
    uint16 b = a; // Implicit conversion: uint8 to uint16 (smaller to larger type)
    uint40 c = a; // Implicit conversion: uint8 to uint40 (smaller to larger type)
    uint64 d = c; // Implicit conversion: uint40 to uint64 (smaller to larger type)
}
```

Implicit conversion - Example

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;

contract Test {
    uint8 private a; // Declare 'a' as a state variable for external access

    // Function to perform type conversions
    function conversion() public {
        a = 10;       // Initialize the state variable
        uint16 b = a; // Implicit conversion: uint8 to uint16
        uint40 c = a; // Implicit conversion: uint8 to uint40
        uint64 d = c; // Implicit conversion: uint40 to uint64
    }

    // Function to check the value of 'a'
    function checkConversion() public view returns (uint8) {
        return a;
    }
}
```

Implicit conversion - Example

```
function conversion1() public returns(int) {
    uint16 a = 10;

    uint8 b = a; // Compile time error
}
function conversion2() public returns(int) {
    int8 a = 10;

    int16 b = a; // NO error
```

Implicit conversion - Example

```
function conversion1() public returns(int) {
    int8 a = 10;

    uint16 b = a; // compile time error
}
function conversion2() public returns(int) {
    uint8 a = 10;
    int16   b = a; // NO error, works fine
}
```

Explicit conversion

Explicit conversion meansconverting larger data type into smaller data type.

Conversion will be not be automatic and external help is required

Explicit conversion

1. Explicit conversion is required when a compiler does not perform implicit conversion either because of loss of data or a value containing data not falling within a target data type range.
2. Solidity provides a **function for each value type** for explicit conversion.
3. Examples of explicit conversion are uint16 conversion to uint8. Data loss is possible in such cases.

Explicit conversion

1. Some of explicit conversion functions are:
    a. int8(), int16(), int24() ...
    b. uint8(), uint16(), uint24() ...
    c. bytes1(), bytes2(), bytes3()
    d. string()
    e. bytes()

Explicit conversion - Example

```
function conversion() public view returns(uint) {
    uint256 a = 10;

    uint16 b = uint16(a); // convert uint256 to uint16
    return b; // 10
}
```

Explicit conversion - Example

```
function conversion() public view returns(uint) {
    uint16 a = 300;
    uint8 b = uint8(a); // data loss
    return b; // 44

}
```

Explicit conversion - Example

```
function conversion() public view returns(int) {
    int16 a = 300;
    int8 b = int8(a); // data loss
    return b; // 44

}
```

Explicit conversion - Example

Return from function

```
function conversion() public view returns(int64){
    int16 b = 50;

    return b; // returns 50
}
```

Explicit conversion - Example

```
    enum Gender {
    male,female,xyz

}
function conversion() public view returns(int) {
    Gender g = Gender.female;
    int16 b = int16(g);
    return b; // 1

}
```

Explicit conversion - Example

```
enum Gender { male,female


}
function conversion() public view returns(Gender) {
    int8 a = 1;
    Gender g = Gender(a);
    return g;  }
```

Explicit conversion - Example

```
function conversion() public {
    string memory name = "Rizwan";
    bytes memory b = bytes(name);

    string memory backToString = string(b);
}
```

Explicit conversion - Example

```
pragma solidity ^0.5.0;

contract First {

    enum Status {
        APPLIED,
        APPROVED,
        LEARNING,
        FAILED,
        GRADUATED
    }
    function convertEnum() public returns (int8) {
        Status s = Status.LEARNING;
        int8 b = int8(s);
    }
}
```

Explicit conversion - Example

```
pragma solidity ^0.5.0;

contract First {

    function convertString() public view returns (bytes memory) {
        string memory a = "Hello World";
        bytes memory b = bytes(a);
        return b;
    }
}
```

Differencebetween tx.origin and msg.sender

1. tx.origin and msg.sender both returns address of transaction sender, but there is difference
2. The tx.origin global variable refers to the original external account that started the transaction, it will always be external account
3. msg.sender refers to the immediate account (it could be external or another contract account) that invokes the function.

Differencebetween tx.origin and msg.sender

4. If there are multiple function invocations on multiple contracts, tx.origin will always refer to the account that **started the transaction** irrespective of the stack of contractsinvoked.
5. However, msg.sender will refer to the **immediate previous account** (contract/external) that invokes the next contract.
6. It is recommended to use msg.sender over tx.origin.

Global val in practice

```
// SPDX-License-Identifier: MIT
pragma solidity ^ 0.8.26;

contract First {
    // Declare events to log various data types
    event logString(string);
    event loguint(uint);
    event logBytes(bytes);
    event logaddress(address);

    // Function to emit global values
    function globalval() public {   infinite gas
        emit loguint(block.gaslimit); // Emit block gas limit
        emit logaddress(msg.sender);  // Emit sender's address
        emit logBytes(msg.data);      // Emit function calldata
    }
}
```

Global val in practice

```
pragma solidity ^0.8.26;

contract First {
    // Declare events to log various data types
    event logString(string);
    event loguint(uint);
    event logBytes(bytes);
    event logaddress(address);

    // Function to emit global values
    function globalval() public {    infinite gas
        emit loguint(block.gaslimit); // Emit block gas limit
        emit logaddress(msg.sender);  // Emit sender's address
        emit logBytes(msg.data);      // Emit function calldata
    }

    // Function to return global values
    function getval() public view returns (string memory, uint, bytes memory, address) {    infinite gas
        return (
            "Global Values",    // Sample string
            block.gaslimit,     // Current block gas limit
            msg.data,           // Input calldata
            msg.sender          // Address of the caller
        );
    }
}
```

THANK-YOU