블록체인
솔리디티 기초

서강대학교
SOGANG UNIVERSITY

목차

- new
- 캡슐화(encapsulation)
- 상속(inheritance)
- assert/revert/require
- try/catch
- modifier
- enum
- Import/라이브러리
- address 자료형

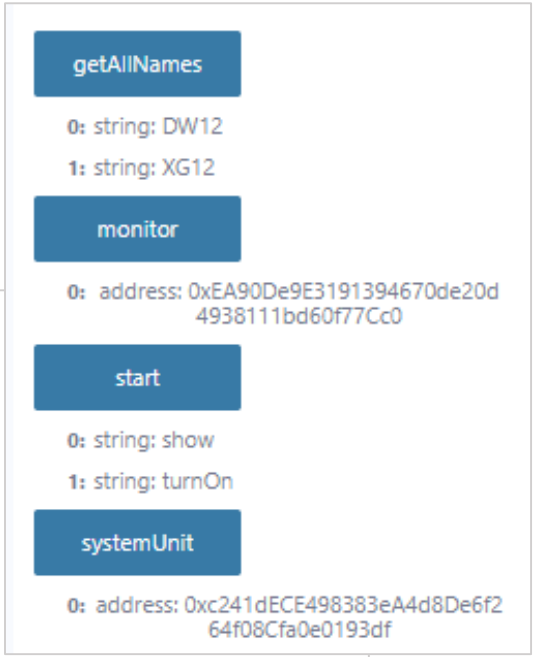
new를 이용한 다른 컨트랙트 생성과 호출(1/2)

- new: 컨트랙트를 새롭게 배포

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.7.0 <0.9.0;

contract Monitor {
    string public name;
    constructor(string memory _name){
        name = _name;
    }
    function show() public pure returns(string memory){
        return "show";
    }
}

contract SystemUnit {
    string public name = "XG12";
    function turnOn() public pure returns(string memory){
        return "turnOn";
    }
}
```

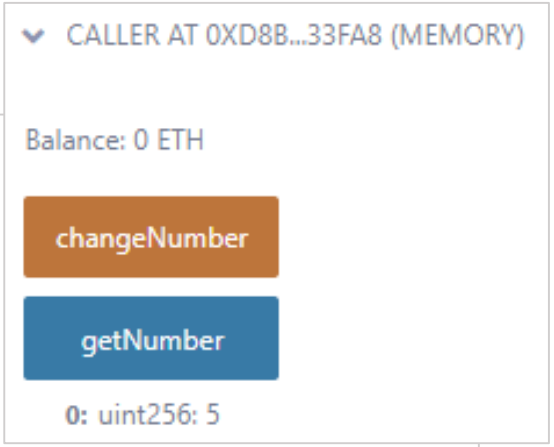
- **new를 이용한 다른 컨트랙트 생성과 호출(2/2)**
  - 점연산자로 다른 컨트랙트의 함수에 접근

```
contract Computer {
    Monitor public monitor;
    SystemUnit public systemUnit;

    constructor(){
        monitor = new Monitor("DW12");
        systemUnit = new SystemUnit();
    }

    function getAllNames() public view returns(string memory,string memory) {
        return(monitor.name(), systemUnit.name());
    }

    function start() public view returns(string memory,string memory) {
        return(monitor.show(), systemUnit.turnOn());
    }
}
```



- **캡슐화(encapsulation)**
  - 제공되는 함수를 통해서만 변수에 접근 가능

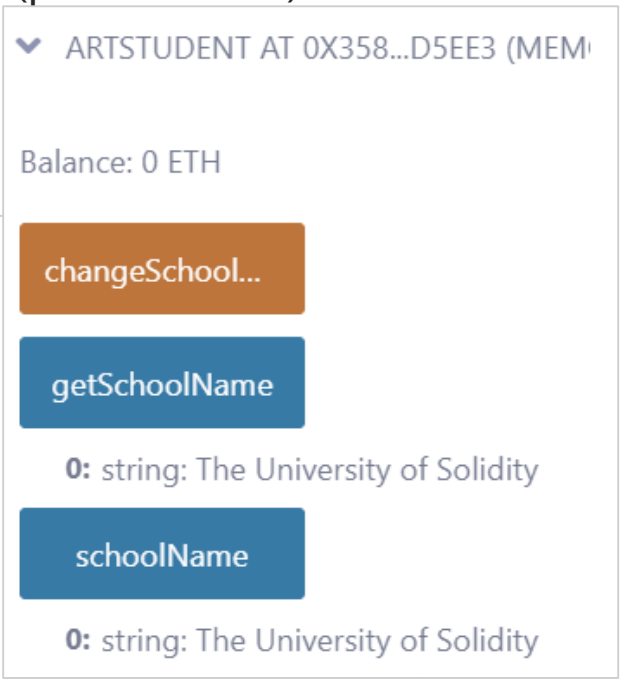
```
contract Number {
    uint private num = 4;
    function changeNum() public {
        num = 5;
    }
    function getNum() public view returns(uint) {
        return num;
    }
}
contract Caller {
    Number internal instance = new Number();
    function changeNumber() public {
        instance.changeNum();
    }
    function getNumber() public view returns(uint) {
        return instance.getNum();
    }
}
```



Solidity 상속

- **상속(inheritance)**
  - 부모 컨트랙트의 변수나 함수를 자식 컨트랙트에서 사용(private 제외)
  - is 를 사용하여 상속받을 컨트랙트 정의
  - 상속 정의

```
contract Student {
    string public schoolName = "The University of Solidity";
}

contract ArtStudent is Student {

    function changeSchoolName() public {
        schoolName = "The University of Blockchain";
    }
    function getSchoolName() public view returns(string memory){
        return schoolName;
    }
}
```



Solidity 상속

- **상속(inheritance): 생성자 매개변수가 있는 부모 컨트랙트 상속**

```
contract Student {
    string public schoolName = "The University of Solidity";
    string public major;
    constructor(string memory _major) {
        major = _major;
    }
}

contract ArtStudent is Student("Art") {
}

contract MusicStudent is Student {
    constructor() Student("Music") {
    }
}

contract MathStudent is Student {
    constructor(string memory _major) Student(_major) {
    }
}
```

Solidity 상속

- **상속(inheritance): private과 internal 변수**
  - private: 정의된 스마트 컨트랙트 내부에서만 접근 가능
  - Internal: 상속받은 스마트 컨트랙트에서도 접근 가능

```
contract Student {
    string private schoolName = "Solidity University";
    string internal schoolNumbers = "02-1234-5678";
}

contract ArtStudent is Student {
    /*
    function getSchoolName() public view returns(string memory){
        return schoolName; // 에러
    }
    */

    function getSchoolNumbers() public view returns(string memory){
        return schoolNumbers;
    }
}
```

```
DeclarationError: Undeclared
identifier.
--> week5.sol:12:16:
|
12 | return schoolName;
| ^^^^^^^^^^
```

ARTSTUDENT AT 0X0FC...9A836 (MEM

Balance: 0 ETH

getSchoolNum...

0: string: 02-1234-5678

Solidity 상속

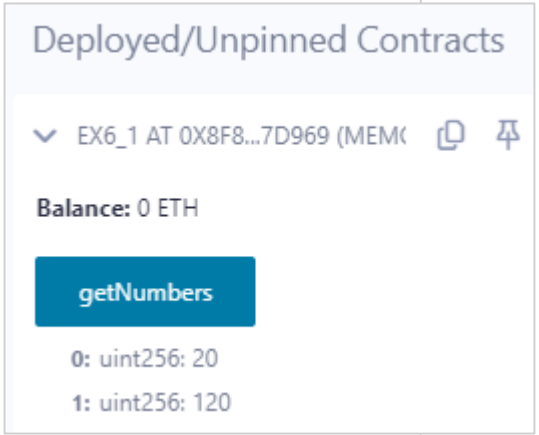
- **Overriding**
  - 자식 컨트랙트에서 부모 컨트랙트의 함수를 재정의(덮어쓰기)하여 다른 동작을 수행하도록 하는 것
  - 부모 컨트랙트 함수: virtual, 자식 컨트랙트 함수: override 사용

```
contract Student {

    function major() public pure virtual returns(string memory) {
        return "Math";
    }
}

contract ArtStudent is Student {

    function major() public pure override returns(string memory) {
        return "Art";
    }
}
```

Solidity Overloading

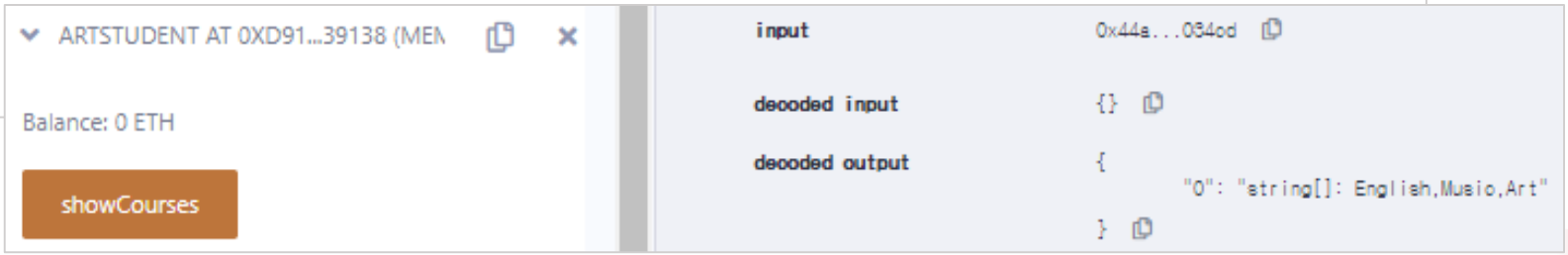
- **Overloading**
  - 함수가 하나의 이름으로 여러 개의 다른 매개변수를 다룰 수 있다

```
contract Calculator {

    function mul(uint _num1, uint _num2) public pure returns(uint) {
        return _num1*_num2;
    }
    function mul(uint _num1, uint _num2, uint _num3) public pure returns(uint) {
        return _num1*_num2*_num3;
    }

}

contract Ex6_1 {

    Calculator internal calculator = new Calculator();
    function getNumbers() public view returns(uint,uint) {
        return (calculator.mul(4,5), calculator.mul(4,5,6));
    }
}
```



- **super: 부모 컨트랙트 함수 호출**

```
contract Student {

    string[] internal courses;
    function showCourses() public virtual returns(string[] memory) {
        delete courses;
        courses.push("English");
        courses.push("Music");
        return courses;
    }
}
contract ArtStudent is Student {

    function showCourses() public override returns(string[] memory) {
        super.showCourses();
        courses.push("Art");
        return courses;
    }
}
```

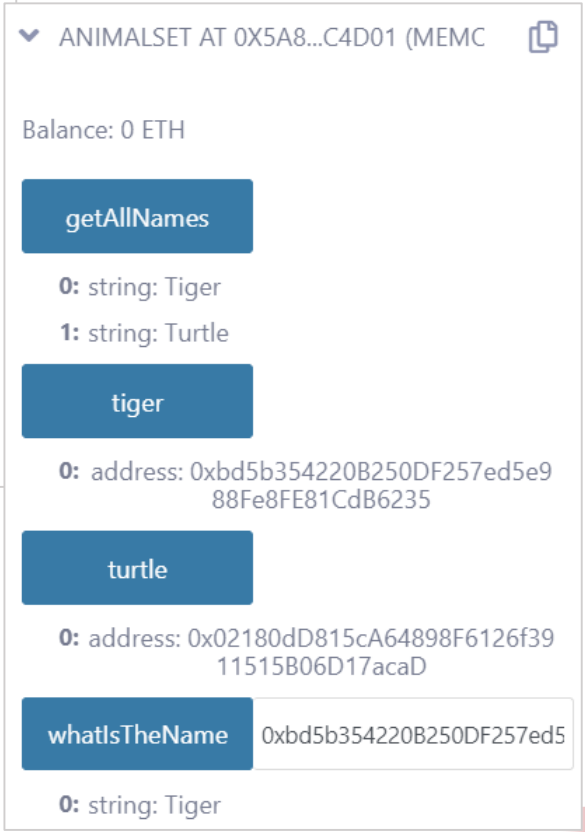


- 부모 스마트 컨트랙트 자료형으로 선언한 인스턴스(1/2)

```
contract Animal {
    function getName() public pure virtual returns(string memory) {
        return "Animal";
    }
}

contract Tiger is Animal {
    function getName() public pure override returns(string memory) {
        return "Tiger";
    }
}

contract Turtle is Animal {
    function getName() public pure override returns(string memory) {
        return "Turtle";
    }
}
```

- 부모 스마트 컨트랙트 자료형으로 선언한 인스턴스(2/2)

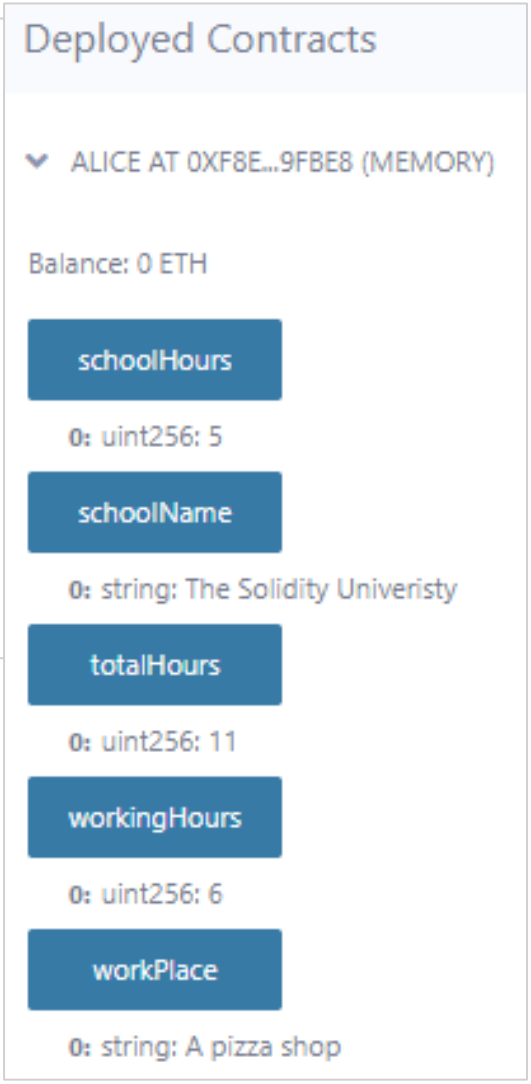
```
contract AnimalSet {
    Animal public tiger = new Tiger();
    Animal public turtle = new Turtle();

    function getAllNames() public view returns(string memory,
string memory) {
        return (tiger.getName(), turtle.getName()) ;
    }

    function whatIsTheName(Animal _animal) public pure
returns(string memory) {
        return (_animal.getName()) ;
    }
}
```



Solidity 상속

- 다중상속: 여러 개의 스마트 컨트랙트 상속

```
contract ArtStudent {
    string public schoolName = "The Solidity Univeristy";
    uint public schoolHours = 5;
}
contract PartTimer {
    string public workPlace = "A pizza shop";
    uint public workingHours = 6;
}

contract Alice is ArtStudent, PartTimer {
    uint public totalHours = schoolHours + workingHours;
}
```



- **다중상속: 함수와 변수 중복 오류**
  - 자식 컨트랙트에 부모 컨트랙트의 함수와 변수를 정의하지 않아야 한다
  - 부모컨트랙트들에 함수와 변수의 중복이 없어야 한다

```
contract ArtStudent {
    uint public times = 7;
    function time() public pure returns(uint) {
        return 3;
    }
}
contract PartTimer {
    function time() public pure returns(uint) {
        return 3;
    }
}

contract Alice is ArtStudent, PartTimer {
    uint public times = 2; // 에러
}
```

```
TypeError: Derived contract must override function "time". Two or more base classes define function with same name and parameter types.
```

```
DeclarationError: Identifier already declared.
```

Solidity 상속

- **추상 컨트랙트: abstract 키워드 사용**
  - 함수의 구현 대신 선언부만을 가지는 컨트랙트 → 상속받은 컨트랙트에서 구현
  - 구현된 함수와 구현되지 않은 함수 모두 포함할 수 있다

```
abstract contract System {
    uint internal version;
    bool internal errorPass;

    function versionCheck() internal virtual;
    function errorCheck() internal virtual;

    function boot() public returns(uint, bool) {
        versionCheck();
        errorCheck();
        return (version, errorPass);
    }
}
```

```
contract Computer is System {
    function versionCheck () internal override {
        version = 3;
    }
    function errorCheck () internal override {
        errorPass = true;
    }
}
contract SmartPhone is System {
    function versionCheck () internal override {
        version = 25;
    }
    function errorCheck () internal override {
        errorPass = false;
    }
}
```

▪ 추상 컨트랙트

COMPUTER AT 0XD7A...F771B (MEMC

Balance: 0 ETH

boot

```
input                0x54d...44bf7

decoded input        {}

decoded output       {
                         "0": "uint256: 3",
                         "1": "bool: true"
                     }
```

SMARTPHONE AT 0XDA0...42B53 (ME

Balance: 0 ETH

boot

```
decoded input        {}

decoded output       {
                         "0": "uint256: 25",
                         "1": "bool: false"
                     }
```

Solidity 상속

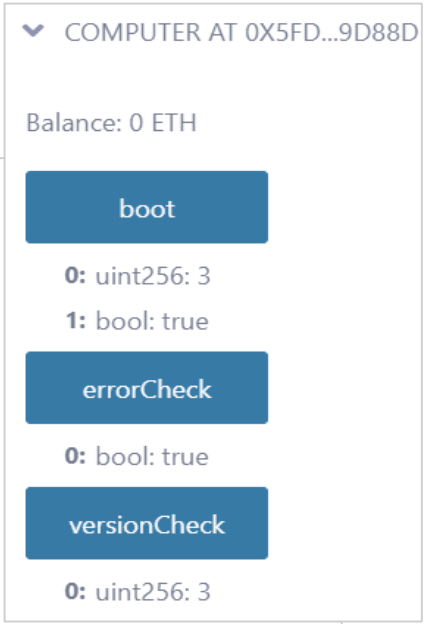
- **인터페이스: interface 키워드 사용**
  - 구현되지 않은 함수만을 갖는다
  - 이미 배포된 컨트랙트를 참조하고 함수를 호출할 때 유용
  - constructor 정의 불가
  - 상태변수 선언 불가
  - 함수의 가시성 지정자는 external이어야 함

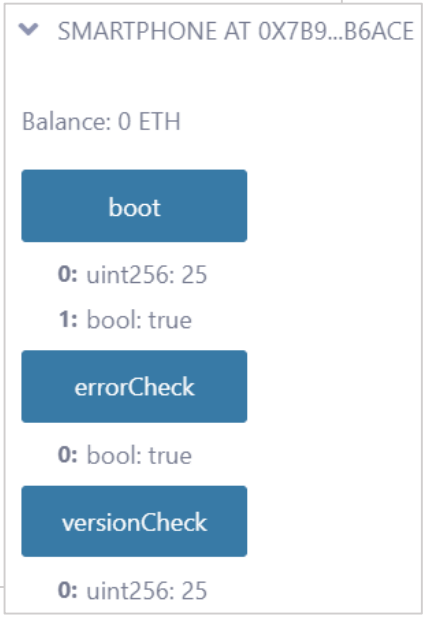
```
interface System {
    function versionCheck() external returns(uint);
    function errorCheck() external returns(bool);
    function boot() external returns(uint, bool);
}
```

Solidity 상속

- 인터페이스

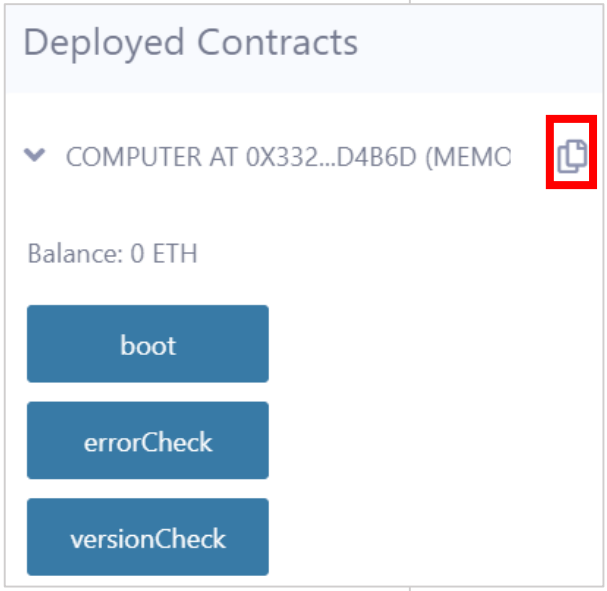
```
contract Computer is System {
    function versionCheck() public pure override returns(uint) {
        return 3;
    }
    function errorCheck() public pure override returns(bool) {
        return true;
    }
    function boot () public pure override returns(uint, bool) {
        return (versionCheck(),errorCheck());
    }
}

contract SmartPhone is System {
    function versionCheck() public pure override returns(uint) {
        return 25;
    }
    function errorCheck() public pure override returns(bool) {
        return true;
    }
    function boot () public pure override returns(uint, bool) {
        return (versionCheck(),errorCheck());
    }
}
```





- 인터페이스를 활용해, 배포된 스마트 컨트랙트 호출
  - 먼저 Computer 배포

```
interface System {
   function versionCheck() external returns(uint);
   function errorCheck() external returns(bool);
   function boot() external returns(uint, bool);
}
contract Computer is System {
   function versionCheck() public pure override returns(uint) {
      return 3;
   }
   function errorCheck() public pure override returns(bool) {
      return true;
   }
   function boot () public pure override returns(uint, bool) {
      return (versionCheck(),errorCheck());
   }
}
```

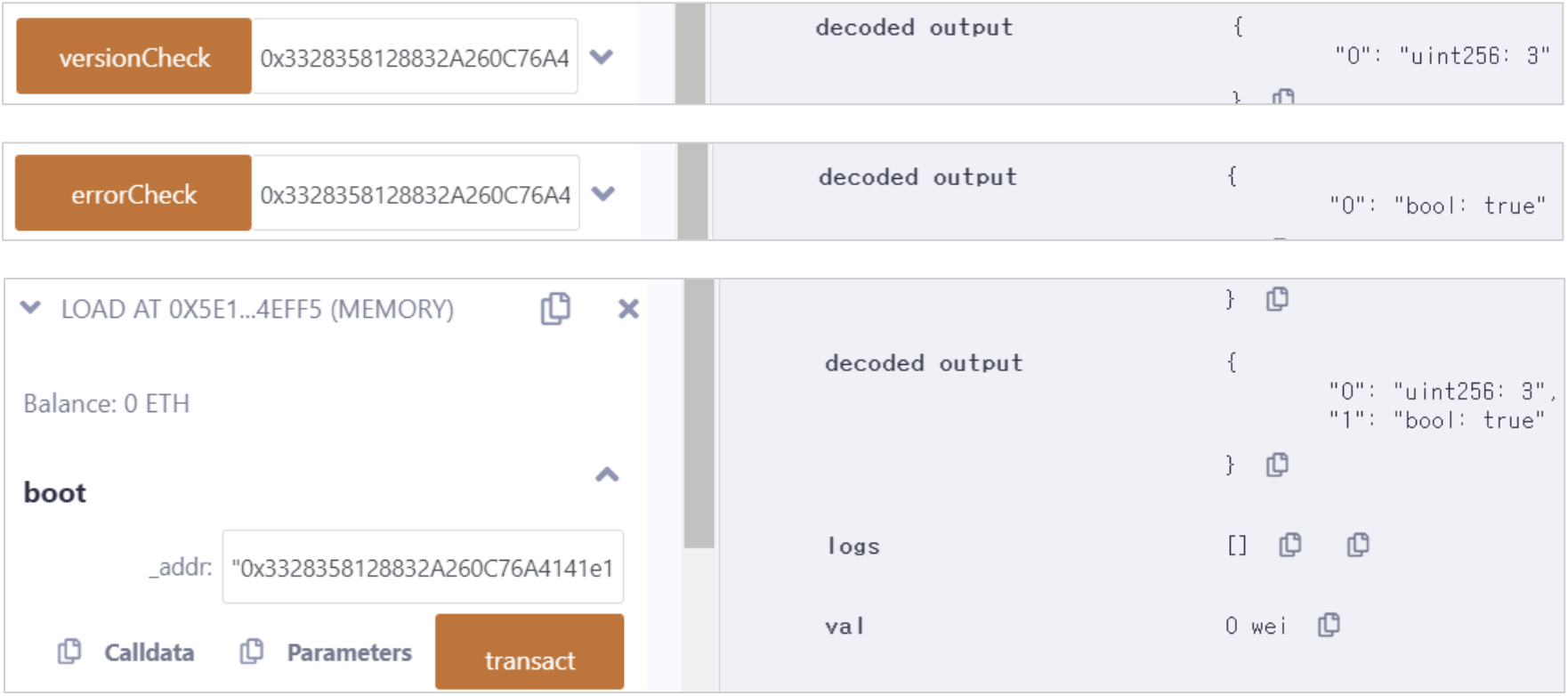


Solidity 상속

- 인터페이스를 활용해, 배포된 스마트 컨트랙트 호출
  - interface명(배포된 스마트 컨트랙트의 주소)

```
interface System {
    function versionCheck() external returns(uint);
    function errorCheck() external returns(bool);
    function boot() external returns(uint, bool);
}

contract Load {
    function versionCheck(address _addr) public returns(uint) {
        return System(_addr).versionCheck();
    }
    function errorCheck(address _addr) public returns(bool) {
        return System(_addr).errorCheck();
    }
    function boot (address _addr) public returns(uint, bool) {
        return System(_addr).boot();
    }
}
```

- 인터페이스를 활용해, 배포된 스마트 컨트랙트 호출



Solidity에서 객체지향

- **캡슐화**
  - 객체의 속성과 메서드를 하나로 묶고, 외부에서 접근을 제한
- **상속**
  - 기존 스마트컨트랙트의 특성을 물려받아 새로운 스마트컨트랙트를 생성
- **다형성**
  - 오버라이딩
  - 오버로딩
- **추상화**
  - 추상 스마트컨트랙트
  - 인터페이스

Solidity(assert/revert/require)

- **assert**
  - 내부적으로 문제가 발생할 때: 0으로 나누기, 존재하지 않는 배열 인덱스 등
  - 불변성 확인: 코드에서 assert 명령문으로 발생시키기

    assert(오류 발생조건이 false이면)

```
assert(userBalance<=totalSupply)
```

- **revert**
  - if 조건문과 같이 사용, 오류 메시지 출력

    revert("오류메시지")

```
if(num <= 10)
   revert("num must be more than 10");
```

- **require**
  - if 조건문과 revert가 합쳐진 것

    require(오류 발생조건이 false이면, "오류메시지")

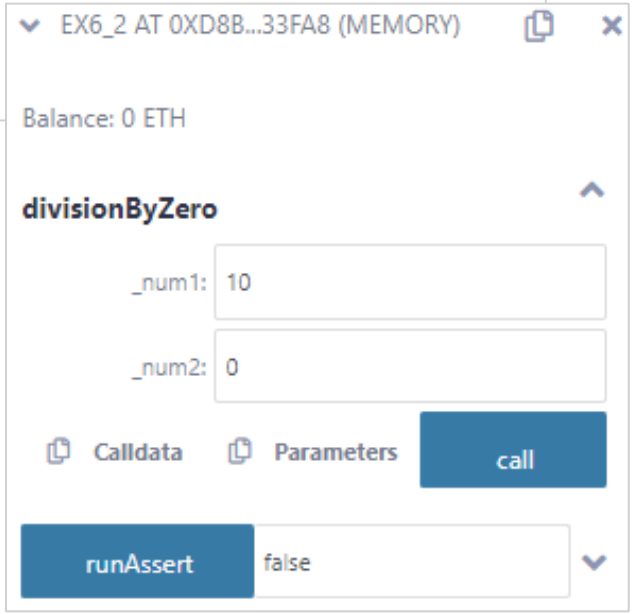
```
require(num>10, "num must be more than 10")
```

Solidity(assert/revert/require)

- **assert**

```
contract Ex6_2 {

    function runAssert(bool _bool) public pure returns(bool) {
        assert(_bool);
        return _bool;
    }
    // 내부 오류 발생
    function divisionByZero(uint _num1, uint _num2) public pure returns(uint)  {
        return _num1/_num2;
    }
}
```

```
CALL   [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Ex6_2.runAssert(bool) data: 0x187...00000

call to Ex6_2.runAssert errored: Error occured: revert.

revert
        The transaction has been reverted to the initial state.
Note: The called function should be payable if you send value and the value you send should be less than your current balance.
Debug the transaction to get more information.

call to Ex6_2.divisionByZero


CAL    [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Ex6_2.divisionByZero(uint256,uint256)
L      data: 0x628...00000
call to Ex6_2.divisionByZero errored: Error occured: revert.

revert
        The transaction has been reverted to the initial state.
Note: The called function should be payable if you send value and the value you send should be less than your current balance.
Debug the transaction to get more information.
```



Solidity(assert/revert/require)

- **revert/require**

```
contract Ex6_3 {
    function runRevert(uint _num) public pure returns(uint) {
        if(_num<=3){
            revert("Revert error : input must be greater than 3");
        }
        return _num;
    }

    function runRequire(uint _num) public pure returns(uint) {
        require(_num>3, "Require error : input must be greater than 3");
        return _num;
    }
}
```

EX6_3 AT 0XF8E...9FBE8 (MEMORY)

Balance: 0 ETH

runRequire	3
runRevert	3

```
CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
     to: Ex6_3.runRequire(uint256) data: 0xcd9...00003
call to Ex6_3.runRequire errored: Error occurred: revert.

revert
        The transaction has been reverted to the initial state.
Reason provided by the contract: "Require error : input must be greater than 3".
You may want to cautiously increase the gas limit if the transaction went out of gas.

call to Ex6_3.runRevert


CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Ex6_3.runRevert(uint256)
     data: 0xa0c...00003
call to Ex6_3.runRevert errored: Error occurred: revert.

revert
        The transaction has been reverted to the initial state.
Reason provided by the contract: "Revert error : input must be greater than 3".
You may want to cautiously increase the gas limit if the transaction went out of gas.
```

- **try/catch: 오류 발생시 처리 구문(1/2)**
  - 오류가 발생하면 catch블록에서 처리

```
contract Ex6_4 {

    event ErrorReason1(string reason);
    event ErrorReason2(uint errorCode);
    event ErrorReason3(bytes lowLevelData);

    function output5(uint _num) public pure returns(uint) {
        if(_num>=6) {
            revert("_num should be 5");
        }
        if(_num<=4){
            assert(false);
        }
        return 5;
    }
```

- **try/catch: 오류 발생시 처리 구문 (2/2)**

```
    function output5WithTryCatch(uint _num) public returns(uint256, bool) {

        try this.output5(_num) returns (uint256 value) {
            return(value, true);
        } catch Error(string memory reason) {
            emit ErrorReason1(reason);
            return(0, false);
        } catch Panic(uint errorCode) {
            emit ErrorReason2(errorCode);
            return(0, false);
        } catch (bytes memory lowLevelData) {
            emit ErrorReason3(lowLevelData);
            return(0, false);
        }
    }
}
```

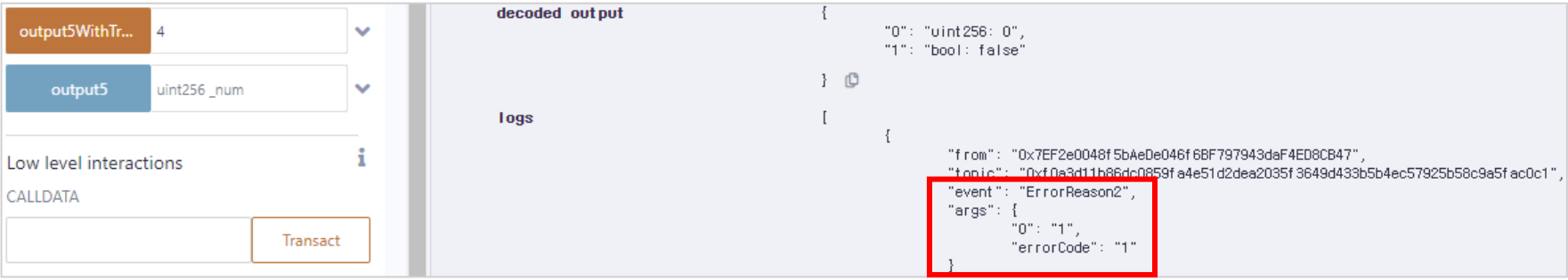
revert/require 오류

assert 오류

그외 나머지 오류

```
TypeError: Try can only be used
with external function calls and
contract creation calls.
--> week6.sol:22:13:
|
22 | try output5(_num) returns
(uint256 value) {
| ^^^^^^^^^^^^
```

- try/catch: 오류 발생시 처리 구문



Panic 에러 코드
https://docs.soliditylang.org/en/latest/control-structures.html#error-handling-assert-require-revert-and-exceptions

- **try/catch: 인스턴스화에 적용**

```
contract Adult {
   uint public age;
   constructor(uint _age) {
      require(_age>19, "Must be more than 19 years old");
      age = _age;
   }
}

contract Ex6_5 {
   event Information(string _error);

   function instantiate(uint _age) public returns(uint) {
      try new Adult(_age) returns(Adult adult) {
         emit Information("Success");
         return(adult.age());
      } catch {
         emit Information("Failed : the default age is 20");
         Adult adult = new Adult(20);
         return(adult.age());
      }
   }
}
```

Solidity 오류 및 예외처리

- **try/catch: 외부함수 호출에 적용**

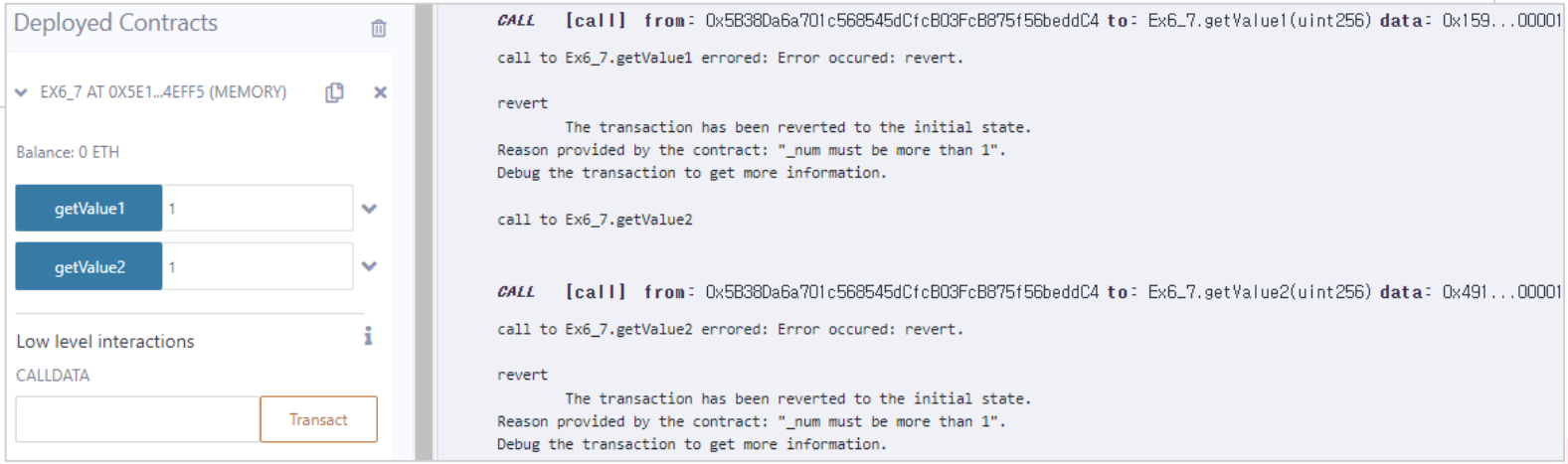
```
contract Math {
    function division(uint _num1, uint _num2) public pure returns(uint)  {
        return _num1/_num2;
    }
}

contract Ex6_6 {
    event Information(string _error);
    Math math = new Math();
    function divisionWithTryCatch(uint _num1, uint _num2) public returns(uint) {
        try math.division(_num1, _num2) returns (uint result) {
            emit Information("Success");
            return(result);
        } catch {
            emit Information("Failure");
            return(0);
        }
    }
}
```

- **modifier**
  - 함수의 공통된 기능을 분리해 모디파이어에 정의하여 대상 함수에 적용
  - 일반적으로 require를 이용해 다수의 함수에 제약을 부여, 재활용
  - 선언

```
// 파라미터 값이 없는 경우
   modifier 모디파이어명{
        revert 나 require
        _;  // modifier가 적용된 함수의 로직 위치
    }

 // 파라미터 값이 있는 경우
    modifier 모디파이어명(파라미터){
        revert 나 require
        _;
    }
```

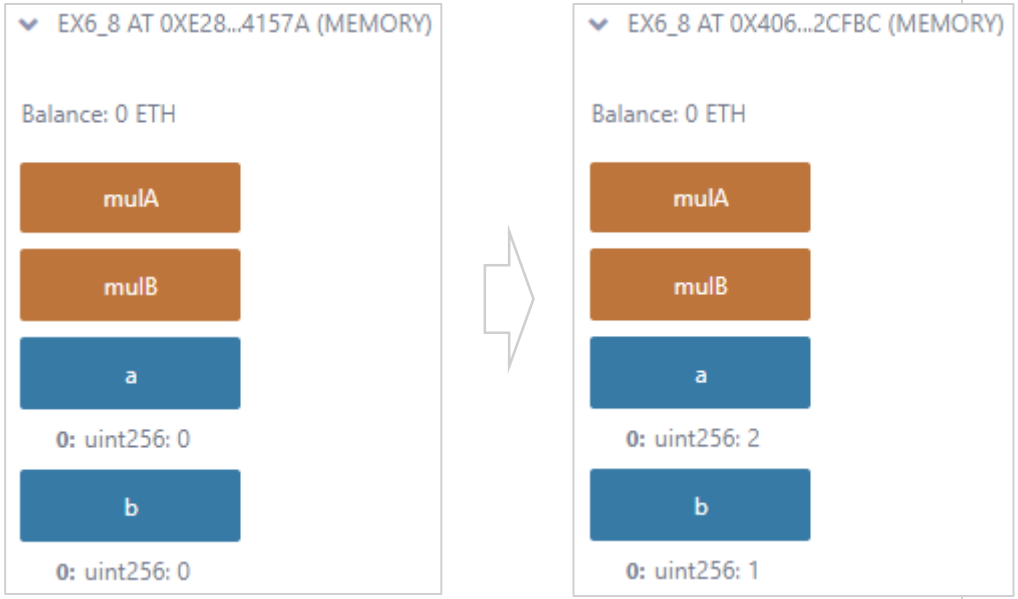
- **modifier**

```
contract Ex6_7 {

    modifier numMoreThan1(uint _num) {
        require(_num>1, "_num must be more than 1");
        _;
    }

    function getValue1(uint _num) public pure numMoreThan1(_num) returns(uint) {
        return _num;
    }

    function getValue2(uint _num) public pure numMoreThan1(_num) returns(uint) {
        return _num*2;
    }

}
```



▪ modifier 실행 순서

```
contract Ex6_8 {

    uint public a;
    uint public b;

    modifier plusA() {
        a = a + 1;
        _;
    }

    modifier plusB() {
        _;
        b = b + 1;
    }

    function mulA() public plusA() {
        a = a * 2;
    }
    function mulB() public plusB() {
        b = b * 2;
    }
}
```



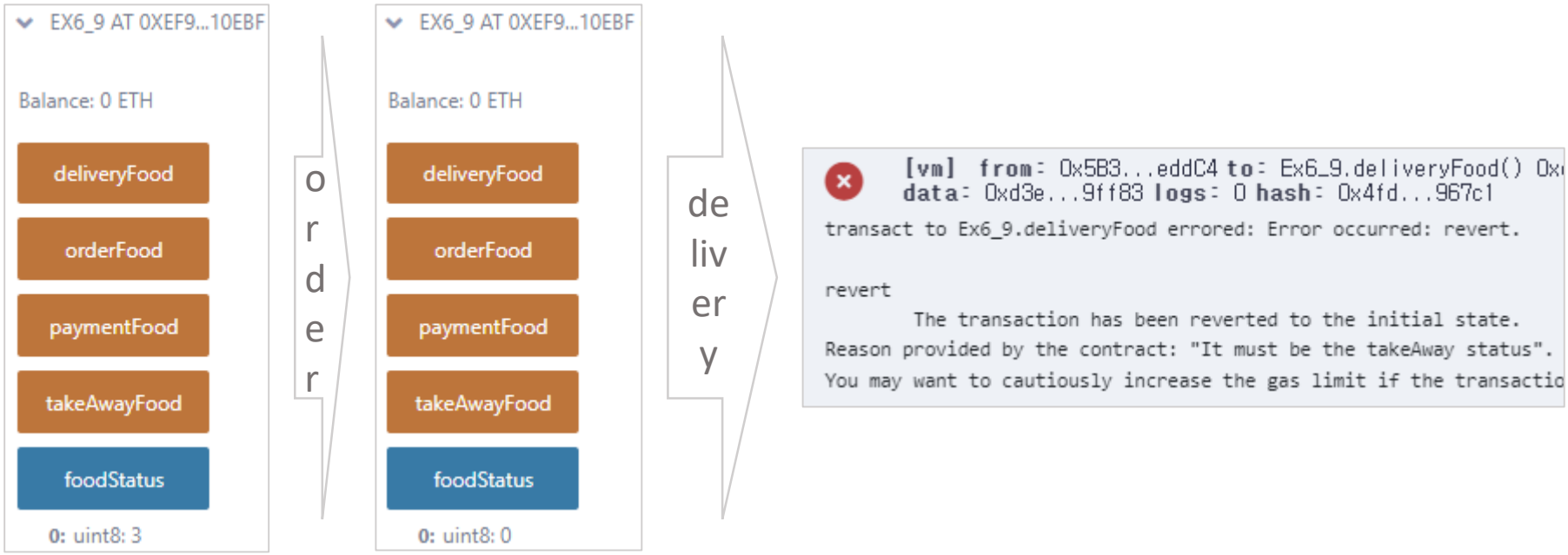
- **enum(열거형)**
  - uint8 범위를 가진 상수집합, 0~255까지 값에 자유롭게 상수명 지정
  - 특정한 값만을 가지는 변수를 만들 때 유용
    - 특정한 과정, 메뉴, 규격 등
    - require 문과 함께 함수의 실행 순서 통제

```
contract Ex6_9 {

    event Information(string info);
    enum FoodProcess {
        order,
        takeAway,
        delivery,
        payment
    }
    FoodProcess public foodStatus;

    constructor(){
        foodStatus = FoodProcess.payment;
    }
```

enum

```
    function orderFood() public {
        require(foodStatus == FoodProcess.payment, "It must be the payment status");
        foodStatus = FoodProcess.order;
        emit Information("Order success");
    }
    function takeAwayFood() public {
        require(foodStatus == FoodProcess.order, "It must be the order status");
        foodStatus = FoodProcess.takeAway;
        emit Information("takeAway success");
    }
    function deliveryFood() public {
        require(foodStatus == FoodProcess.takeAway, "It must be the takeAway status");
        foodStatus = FoodProcess.delivery;
        emit Information("delivery success");
    }
    function paymentFood() public {
        require(foodStatus == FoodProcess(2), "It must be the delivery status");
        foodStatus = FoodProcess.payment;
        emit Information("payment success");
    }
}
```

- **enum**




```
logs
[
    {
        "from": "0xEf9f1ACE83dfbB8f559Da621f4aEA72C6EB10eBf",
        "topic": "0xff8fea957fb7cb582ef297c1c74436fe7a9914efbc52adf197600dc9d171efbc",
        "event": "Information",
        "args": {
            "0": "Order success",
            "info": "Order success"
        }
```

- **Import**
  - 여러 파일에 나누어 작성된 코드를 재사용하거나 모듈화하기 위해
  - 외부 파일 불러오기

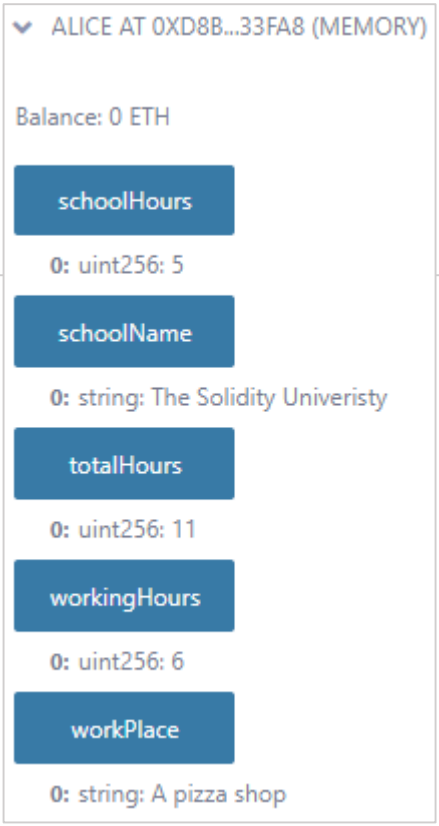
```
//SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.7.0 <0.9.0;

import "./Ex6_10_1.sol";
import "./in/Ex6_10_2.sol";

contract Alice is ArtStudent, PartTimer {
    uint public totalHours = schoolHours + workingHours;
}
```

  - Ex6_10_1.sol

```
//SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.7.0 <0.9.0;

contract ArtStudent {
    string public schoolName = "The Solidity Univeristy";
    uint public schoolHours = 5;
}
```

- **import**
  - in/Ex6_10_2.sol

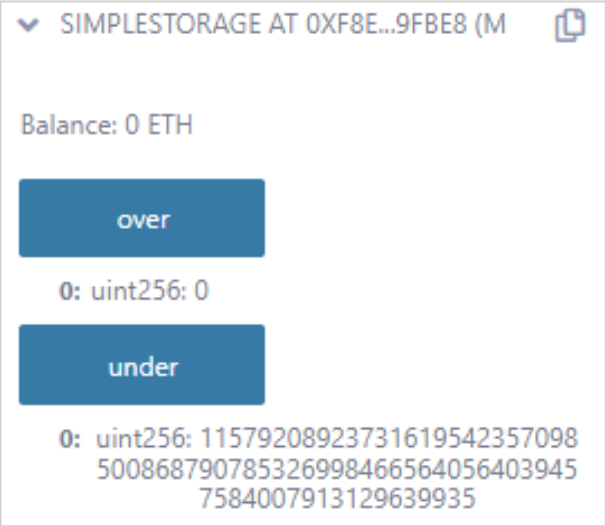
```
//SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.7.0 <0.9.0;

contract PartTimer {
    string public workPlace = "A pizza shop";
    uint public workingHours = 6;
}
```



Solidity 라이브러리(참고)

- **overflow와 underflow**

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.7.6;

contract SimpleStorage {
    uint min = 0;
    uint max = 115792089237316195423570985008687907853269984665640564039457584007913129639935;

    function under() public view returns( uint) {
        return min - 1;
    }

    function over() public view returns( uint) {
        return max + 1;
    }
}
```



- 라이브러리
  - 다른 스마트 컨트랙트에서 재사용할 수 있는 코드 블록
    - 공통 기능을 다른 스마트컨트랙트에서 중복 구현하지 않고도 코드를 간결하게 유지

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.5.0 < 0.8.0;

library Math{
    function add(uint8 a, uint8 b) internal pure returns (uint8) {
        require(a+b >= a , "Error: addition overflow");
        return a+b;
    }
}
```

```
contract Ex6_11{
    using Math for uint8;

    function overflow(uint8 _num1,uint8 _num2) public pure returns(uint8) {
        return _num1+_num2;
    }
    function noOverflow1(uint8 _num1,uint8 _num2) public pure returns(uint8) {
       return Math.add(_num1 ,_num2);
    }
    function noOverflow2(uint8 _num1,uint8 _num2) public pure returns(uint8) {
       return _num1.add(_num2);
    }
}
```

- 라이브러리
  - 결과

overflow

_num1: 255

_num2: 1

Calldata Parameters call

0: uint8: 0

noOverflow1

_num1: 255

_num2: 1

Calldata Parameters call

noOverflow2

_num1: 255

_num2: 1

Calldata Parameters call

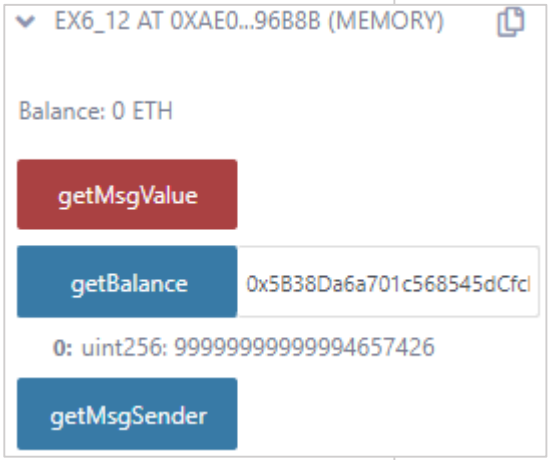
```
call to Ex6_11.noOverflow1

CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
     to: Ex6_11.noOverflow1(uint8,uint8) data: 0xb71...00001
call to Ex6_11.noOverflow1 errored: Error occurred: revert.

revert
        The transaction has been reverted to the initial state.
Reason provided by the contract: "Error: addition overflow".
You may want to cautiously increase the gas limit if the transaction went out of gas.

call to Ex6_11.noOverflow2

CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
     to: Ex6_11.noOverflow2(uint8,uint8) data: 0x508...00001
call to Ex6_11.noOverflow2 errored: Error occurred: revert.

revert
        The transaction has been reverted to the initial state.
Reason provided by the contract: "Error: addition overflow".
You may want to cautiously increase the gas limit if the transaction went out of gas.
```

- **address 자료형**
  - 20바이트, 16진수로 표현
  - EOA(외부소유계정), CA(컨트랙트 계정) 주소 저장
  - (address).balance: 현재 주소가 보유한 이더 잔액을 wei 단위로 표시(uint)

- **payable 키워드**
  - address 자료형과 함수에 사용 가능
  - 이더 수신 가능

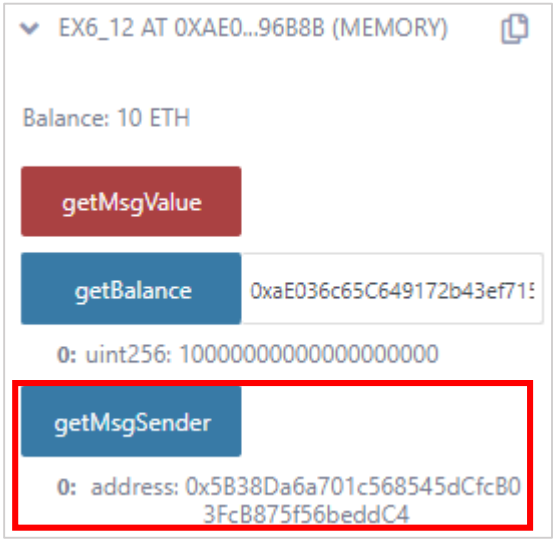
- **전역변수**
  - 블록 정보등 블록체인의 전반적인 정보
  - msg.sender: address 자료형, 현재 메시지 호출자(트랜잭션 발생시킨 발신자)
  - msg.value: uint형, 메시지와 함께 전송된 이더(wei 단위)

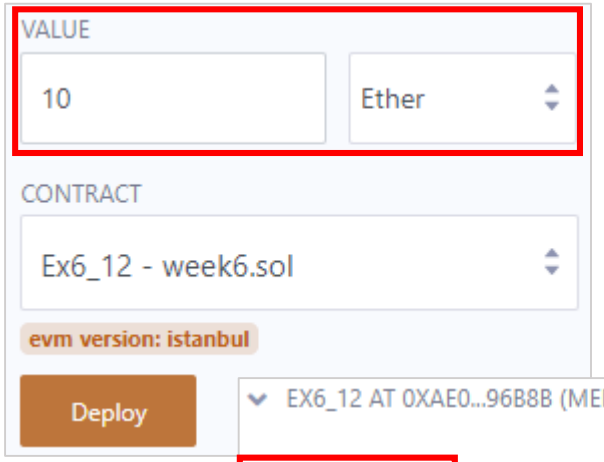
```
contract Ex6_12 {

  function getBalance(address _address) public view returns(uint) {
    return _address.balance;
  }

  function getMsgValue() public payable returns(uint) {
    return msg.value;
  }

  function getMsgSender() public view returns(address) {
    return msg.sender;
  }
}
```



전역변수 https://docs.soliditylang.org/en/latest/units-and-global-variables.html

- **address 자료형, payable 키워드, 전역변수**
  - Remix에서 이더 송금
    - getMsgValue() 호출하면서 이더 송금





- **전역변수: msg.sender 응용(배포자만 특정 함수 실행 가능하게)**
  - 배포자가 아닌 계정은 modifier onlyOwner()가 적용된 함수 호출 불가

```
contract Ex6_13 {

    address public owner;
    modifier onlyOwner() {
        require(owner == msg.sender, "Error: caller is not the owner");
        _;
    }
    constructor () {
        owner = msg.sender; // 배포자 주소 저장
    }

    function getBalance(address _address) public view onlyOwner() returns(uint) {
        return _address.balance;
    }

    function getMsgValue() public payable onlyOwner returns(uint) {
        return msg.value;
    }
}
```