PIEMONTE
FONDI STRUTTURALI E DI INVESTIMENTO EUROPEI 2014/2020
fondo sociale europeo
ITS TECNOLOGIE DELL'INFORMAZIONE E DELLA COMUNICAZIONE

Smart contract e Solidity

Corso Fintech Software Developer

Modulo **Tecnologie Blockchain**

Docente: Dott. Enrico Zimuel

in collaborazione con:
MIUR
UNIONE EUROPEA
REGIONE PIEMONTE
per una crescita intelligente, sostenibile ed inclusiva
www.regione.piemonte.it/europa2020
INIZIATIVA CO-FINANZIATA CON FSE

Sommario

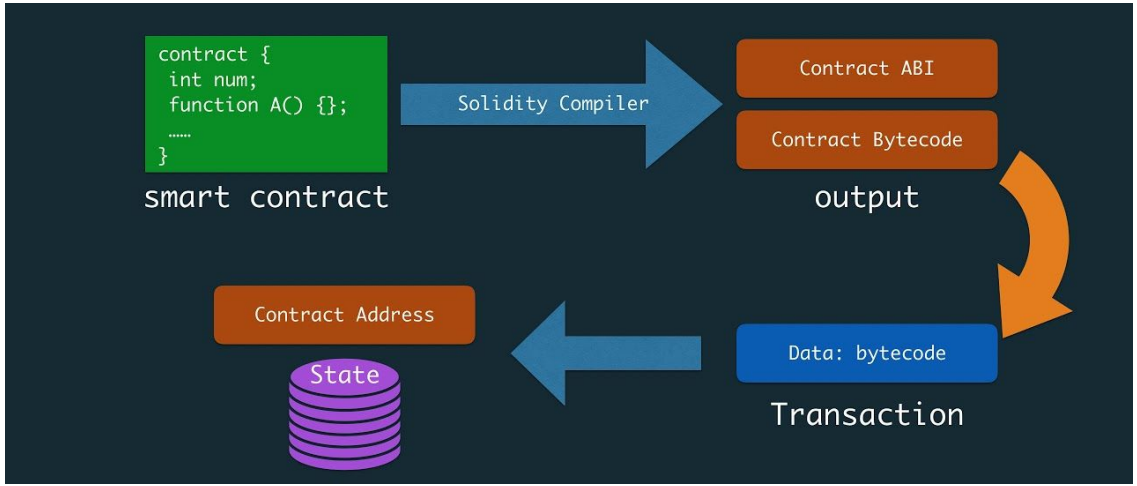
- Introduzione a Solidity
- Solidity e smart contract
- Esempi: Storage, Coin
- Costrutti di base del linguaggio
- Funzioni modifier
- Struct
- Laboratorio

Solidity

- Solidity è un linguaggio a oggetti con **tipizzazione statica** e una sintassi con **parentesi graffe** (curly-braces) ideato per sviluppare smart contract per Ethereum
- Ideato nel 2014 da Gavin Wood
- Quando si sviluppa in Solidity è necessario utilizzare sempre l'ultima versione, poiché soltanto questa riceve security fixes (tranne casi eccezionali).
- Attualmente, l'ultima versione è la 0.8.19

Solidity e smart contract

- Per Solidity un contratto (smart contract) è una collezione di **codice** (funzioni) e **dati** (stati) che risiedono ad uno specifico indirizzo della blockchain Ethereum



Esempio: storage

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.4.16 <0.9.0;

contract SimpleStorage {
    uint storedData;

    function set(uint x) public {
        storedData = x;
    }

    function get() public view returns (uint) {
        return storedData;
    }
}
```

variabile di stato
unsigned integer (256 bits)

Esempio: coin

```sol
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.4;

contract Coin {
    // The keyword "public" makes variables
    // accessible from other contracts
    address public minter;
    mapping (address => uint) public balances;

    // Events allow clients to react to specific
    // contract changes you declare
    event Sent(address from, address to, uint amount);

    // Constructor code is only run when the contract
    // is created
    constructor() {
        minter = msg.sender;
    }

    // Sends an amount of newly created coins to an address
    // Can only be called by the contract creator
    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);
        balances[receiver] += amount;
    }

    // Errors allow you to provide information about
    // why an operation failed. They are returned
    // to the caller of the function.
    error InsufficientBalance(uint requested, uint available);

    // Sends an amount of existing coins
    // from any caller to an address
    function send(address receiver, uint amount) public {
        if (amount > balances[msg.sender])
            revert InsufficientBalance({
                requested: amount,
                available: balances[msg.sender]
            });

        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }
}
```

Costrutti di base del linguaggio

- In Solidity ci sono i costrutti **if**, **else**, **while**, **do**, **for**, **break**, **continue**, **return** così come in C o Javascript
- Le variabili di stato possono essere:
  - **public**, visibili da altri contratti esterni
  - **internal,** visibili solo all'interno del contratto e da tutti quelli che lo estendono
  - **private,** visibili solo all'interno del contratto e non da contratti derivati
- Anche le funzioni possono essere public, internal o private. Inoltre possono essere anche **external** ossia eseguibili da altri smart contract
- Gestione delle eccezioni con **try/catch** ma solo per funzioni esterne

Funzioni modifier

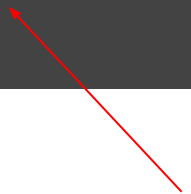
- I **modifier** sono delle funzioni speciali di Solidity che consentono di eseguire delle condizioni prima della loro esecuzione
- Tecnicamente modificano il comportamento di una funzione con una modalità dichiarativa

Esempio di modifier

```
contract owned {
    constructor() { owner = payable(msg.sender); }
    address payable owner;

    modifier onlyOwner {
        require(
            msg.sender == owner,
            "Only owner can call this function."
        );
        _;
    }
}
```

```
contract destructible is owned {
    function destroy() public onlyOwner {
        selfdestruct(owner);
    }
}
```



body della funzione

Struct

- E’ possibile raggruppare più variabili in una stessa struttura (struct)
- Le **struct** sono presenti in altri linguaggi, es. C

```
contract test {
  struct Book {
     string title;
     string author;
     uint book_id;
  }
  Book book;

  function setBook() public {
     book = Book('Learn Java', 'TP', 1);
  }
  function getBookId() public view returns (uint) {
     return book.book_id;
  }
}
```

Laboratorio

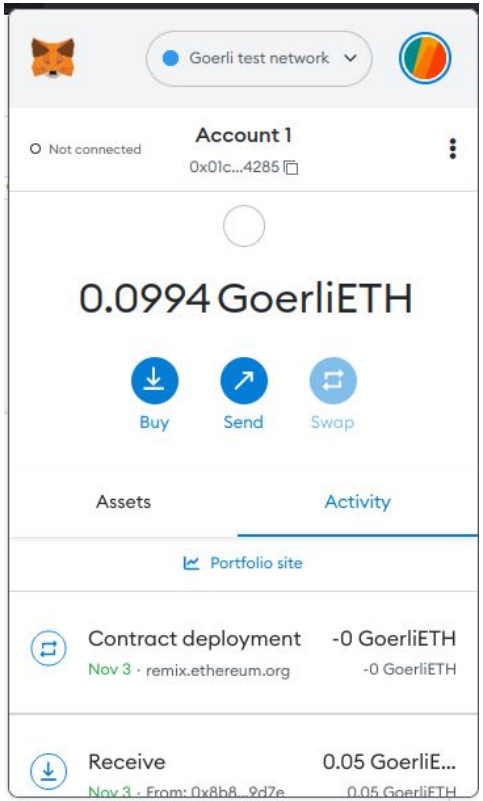
- Installazione di un wallet, es. MetaMask
- Utilizzo della rete di test Goerli
- Richiesta di ETH su Goerli
- Utilizzo dell'IDE Remix
- Esercizi

MetaMask

- **MetaMask** è un wallet che gira come estensione di un browser (eg. Chrome, Firefox, etc) o come app iOS o Android
- Siti web o dapp possono collegarsi, autenticarsi e/o eseguire funzionalità di smart contract utilizzando il wallet di MetaMask
- Installazione su https://metamask.io/download/

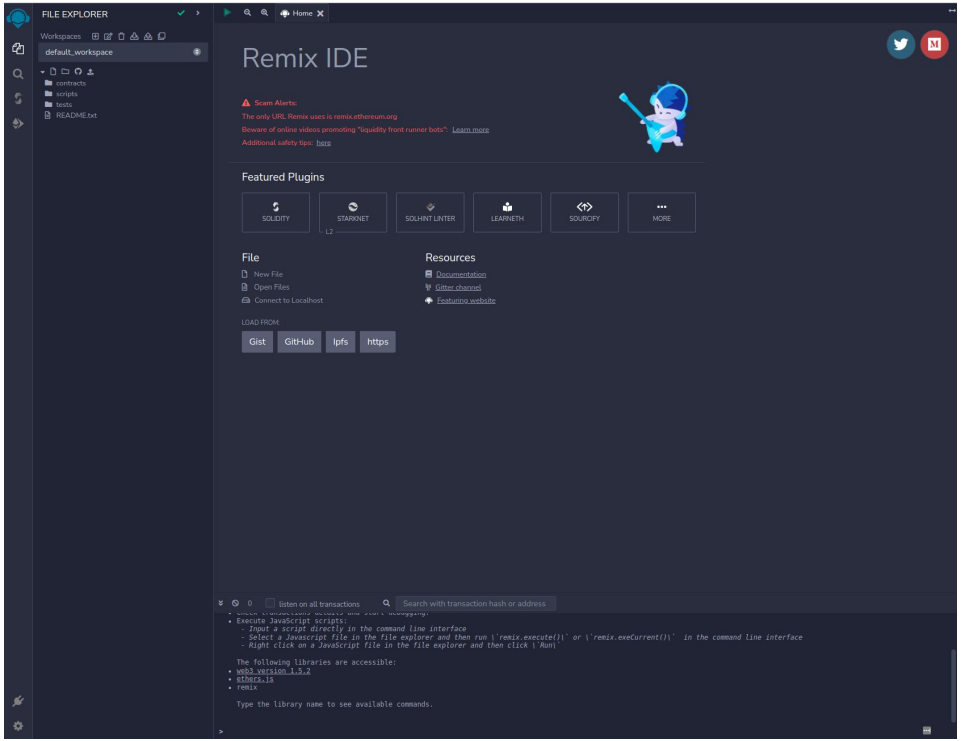
Test network Goerli

- Per poter eseguire degli smart contract è necessario prima testarli su un network di test
- Esistono diversi network test come Goerli ETH
- E’ possibile richiedere ETH di test utilizzando un servizio Goerli Faucet



IDE Remix

- Remix è un progetto contenente diversi tool (IDE, CLI, VS code plugin) per lo sviluppo di smart contract su Ethereum
- E' disponibile un IDE online all'indirizzo https://remix.ethereum.org/



Esercizio 1

- Provare a compilare lo smart contract SimpleStorage con REMIX, qual'è l'opcode generato? Che dimensione ha l'opcode generato?
- Provare ad eseguire lo smart contract memorizzando (set) il numero 42
- Provare ad eseguire la lettura (get) e verificare che il numero restituito sia 42
- Fare il deploy su Goerli testnet utilizzando MetaMask (Environment: Injected Provider MetaMask su REMIX)
- Eseguire la scritture (set) su Goerli, qual'è la fee della transazione?
- Eseguire la lettura (get) su Goerli, qual'è la fee della transazione?

Esercizio 2

- Provare a compilare lo smart contract Coin con REMIX, qual'è l'opcode generato? Che dimensione ha l'opcode generato?
- Fare il deploy su Goerli testnet utilizzando MetaMask (Environment: Injected Provider MetaMask su REMIX)
- Provare ad eseguire lo smart contract per minare delle monete (mint)
- Provare ad inviare delle monete (send) all'indirizzo di un vs. collega

Esercizio 3

- Provare a studiare lo smart contract Voting
- Provare a compilare ed eseguire lo smart contract sulla rete di test di REMIX VM (London o Berlin)
- Fare il deploy dello smart contract sulla rete Goerli testnet

Esercizio 4

- Provare a studiare lo smart contract SimpleAuction
- Provare a compilare ed eseguire lo smart contract sulla rete di test di REMIX VM (London o Berlin)
- Fare il deploy dello smart contract sulla rete Goerli testnet

Esercizio 5

- Provare a studiare lo smart contract Purchase
- Provare a compilare ed eseguire lo smart contract sulla rete di test di REMIX VM (London o Berlin)
- Fare il deploy dello smart contract sulla rete Goerli testnet

Esercizio 6

- Studiare la dapp Micropayment Channel che utilizza del codice javascript con la libreria web3.js e gli smart contract ReceiverPays e SimplePaymentChannel per implementare un servizio di micropagamenti su Ethereum

Riferimenti

- Andreas Antonopoulos, Gavin Wood, Mastering Ethereum: Building Smart Contracts and Dapps, O'Reilly, 2018
- Solidity documentazione ufficiale
- Kevin Solorio, Randall Kanna, David H. Hoover, Hands-On Smart Contract Development with Solidity and Ethereum: From Fundamentals to Deployment, O'Reilly, 2019
- Ritesh Modi, Solidity Programming Essentials: A beginner's guide to build smart contracts for Ethereum and blockchain, Packt, 2018
- Jitendra Chittoda, Mastering Blockchain Programming with Solidity, Packt, 2019
- Solidity Tutorial - A Full Course on Ethereum, Blockchain Development, Smart Contracts, and the EVM, Youtube Video

Grazie dell’attenzione!

Per informazioni:
enrico.zimuel@its-ictpiemonte.it

POR Piemonte
FSE 2014-2020