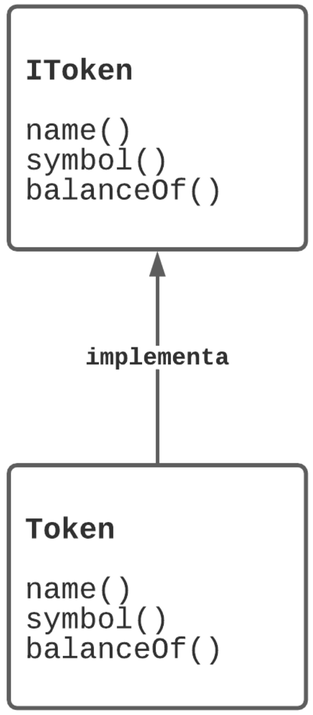
📖 Solidity: Scriviamo un ERC20

- Interfacce
- Eventi
- Underflow, overflow
- Ethereum Improvement Proposals (EIPs)
- EIP-20: Token Standard
- Interfaccia di un ERC20
- Hardhat: setup e scrittura di un contratto
- Q&A

Interfacce

Le interfacce permettono di interagire con altri contratti senza dover dichiarare la loro implementazione.

Le interfacce possono definire inoltre i metodi che devono essere implementati da un contratto.



Eventi

In solidity è possibile dichiarare e emettere eventi.

Gli eventi utilizzano il campo *logs* di una transazione per astrarre ciò che accade durante l'esecuzione della transazione.

Possono essere dichiarati in un contratto o in un interfaccia e sono ereditati dai contratti che derivano da essi.

Le applicazioni possono ascoltare eventi o analizzarli a partire da una transazione.

```
event Event(address a, uint256 x);
```

Underflow, overflow

Le operazioni aritmetiche (+ - / *) possono fallire in caso di overflow/underflow.

Prima della release di solidity 0.8 le operazioni non erano controllate e una operazione che risultava in un over/under-flow eseguiva il wrapping del valore (eg. uint256 x = uint256(-1) valorizzava x come 2^256 - 1).

Con solidity 0.8 è possibile avere il wrapping utilizzando un blocco unchecked.

```
unchecked {
    uint256 x = uint256(-1) // x = 2^256 - 1
}
```

Ethereum Improvement Proposals

Le **Ethereum Improvement Proposals** (EIP) descrivono degli standard per la piattaforma Ethereum e possono essere di vari tipi:

- **Core**: proposte per dei cambiamenti che impattano il protocollo e che necessitano una hard fork
- **Networking**: proposte per il networking dei client (devp2p, Light Ethereum Subprotocol)
- **ERC**: proposte di standard e convenzioni al livello application come token, name registries, librerie e formati per i wallet.

EIP-20: Token Standard

EIP-20 propone una API standard per la gestione di *token* da parte di smart contract ed EOA.

Le funzioni di base descritte da questa proposal sono il **trasferimento** e l’**approvazione** di terze parti ad eseguire trasferimenti.

EIP-20: Token Standard

I metodi esposti (alcuni opzionali) da un token ERC20 sono:

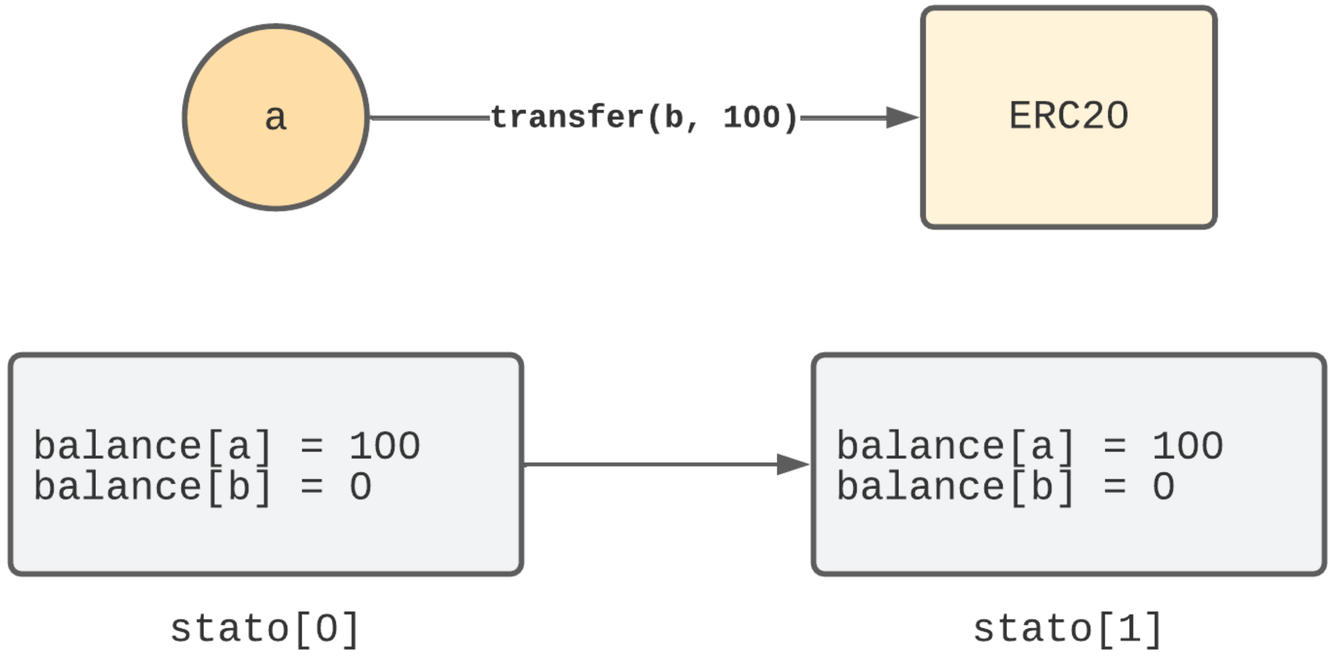
- name(): il nome del token (opzione)
- symbol(): il simbolo che identifica il token (opzionale)
- decimals(): il numero di decimali (opzionale)
- totalSupply(): il numero di token in circolazione
- balanceOf(address *account*): il numero di token posseduti da un utente
- transfer(address *to*, uint256 *amount*): trasferisce i token di un utente ad un altro indirizzo
- transferFrom(address *from*, address *to*, uint256 *amount*): trasferisce i token di un utente ad un altro indirizzo e può essere chiamata da un indirizzo differente dal proprietario
- approve(address *spender*, uint256 *amount*): approva una quantità di token da spendere ad un altro indirizzo
- allowance(address *owner*, address *spender*): mostra la quantità di token appartenenti ad *owner* approvati per essere spesi da *spender*

EIP-20: Token Standard

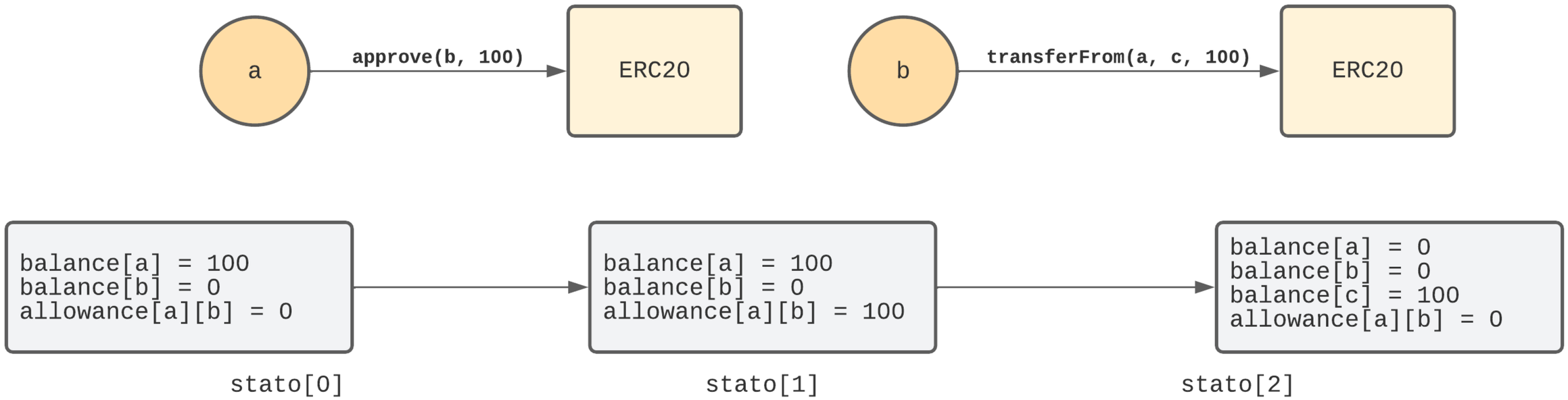
I metodi che effettuano delle modifiche allo stato del contratto devono ritornare un `bool` che indica lo stato della modifica.

Inoltre lo standard EIP-20 indica due eventi: `Transfer` e `Approval`

```
transfer(address to, uint256 amount)
```



approve(address *spender*, uint256 *amount*)



Implementazione di un ERC20: storage

```
string public name;
string public symbol;
uint8 public decimals;

uint256 totalSupply;
mapping(address => uint256) public balanceOf;
mapping(address => mapping(address => uint256)) public allowance;
```

Implementazione di un ERC20: eventi

```
event Transfer(address from, address to, uint256 amount);

event Approval(address owner, address spender, uint256 amount);
```

Implementazione di un ERC20: transfer(address *to*, uint256 *amount*)

```
function transfer(address to, uint256 amount) external returns(bool) {
    balanceOf[msg.sender] -= amount;
    balanceOf[to] += amount;

    emit Transfer(msg.sender, to, amount);

    return true;
}
```

Implementazione di un ERC20: approve(address *spender*, uint256 *amount*)

```
function approve(address spender, uint256 amount) external returns(bool) {
    allowance[msg.sender][spender] = amount;

    emit Approval(msg.sender, to, amount);

    return true;
}
```

Implementazione di un ERC20: transferFrom(address *from*, address *to*, uint256 *amount*)

```
function transferFrom(
    address from,
    address to,
    uint256 amount
) external returns(bool) {
    uint256 allowed = allowance[from][msg.sender];
    allowance[from][msg.sender] = allowed - amount;

    balanceOf[from] -= amount;
    balanceOf[to] += amount;

    emit Transfer(from, to, amount);

    return true;
}
```