ERC-20, OPEN ZEPLIN

What are custom tokens?

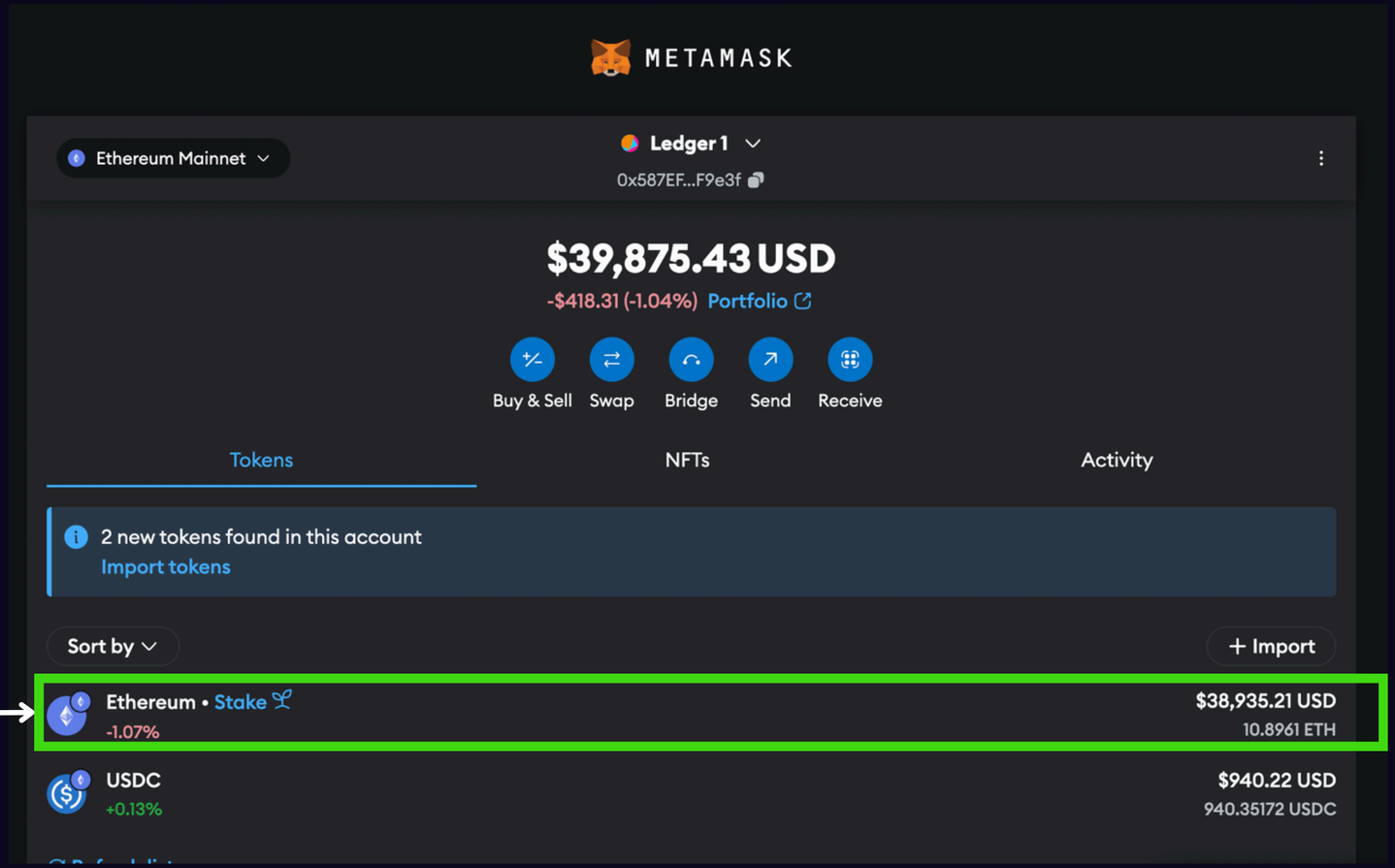
What are custom tokens?

Any token other than native
ETH on the eth blockchain
is a custom token

What are custom tokens?

Any token other than native ETH on the eth blockchain is a custom token

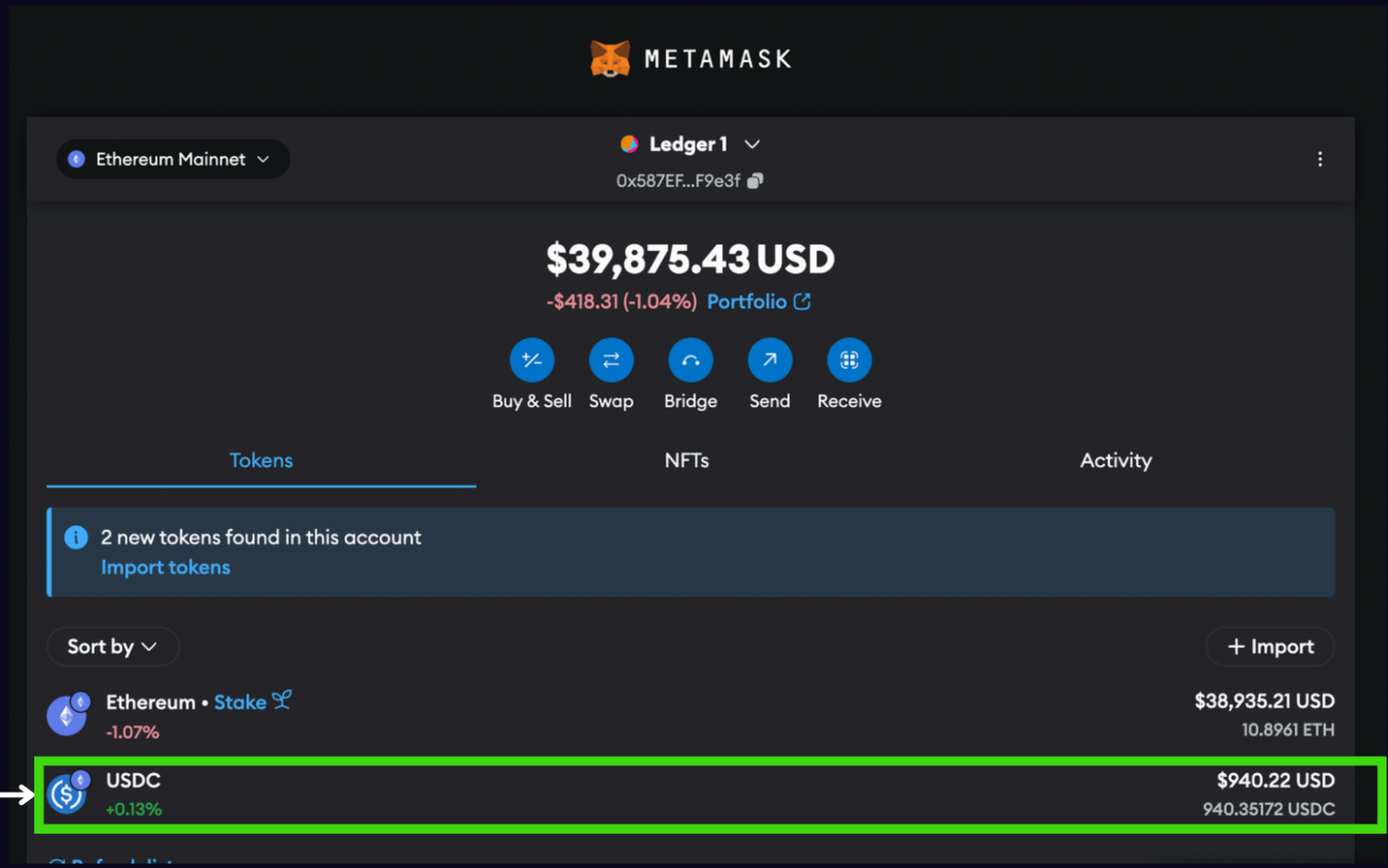
Native ETH



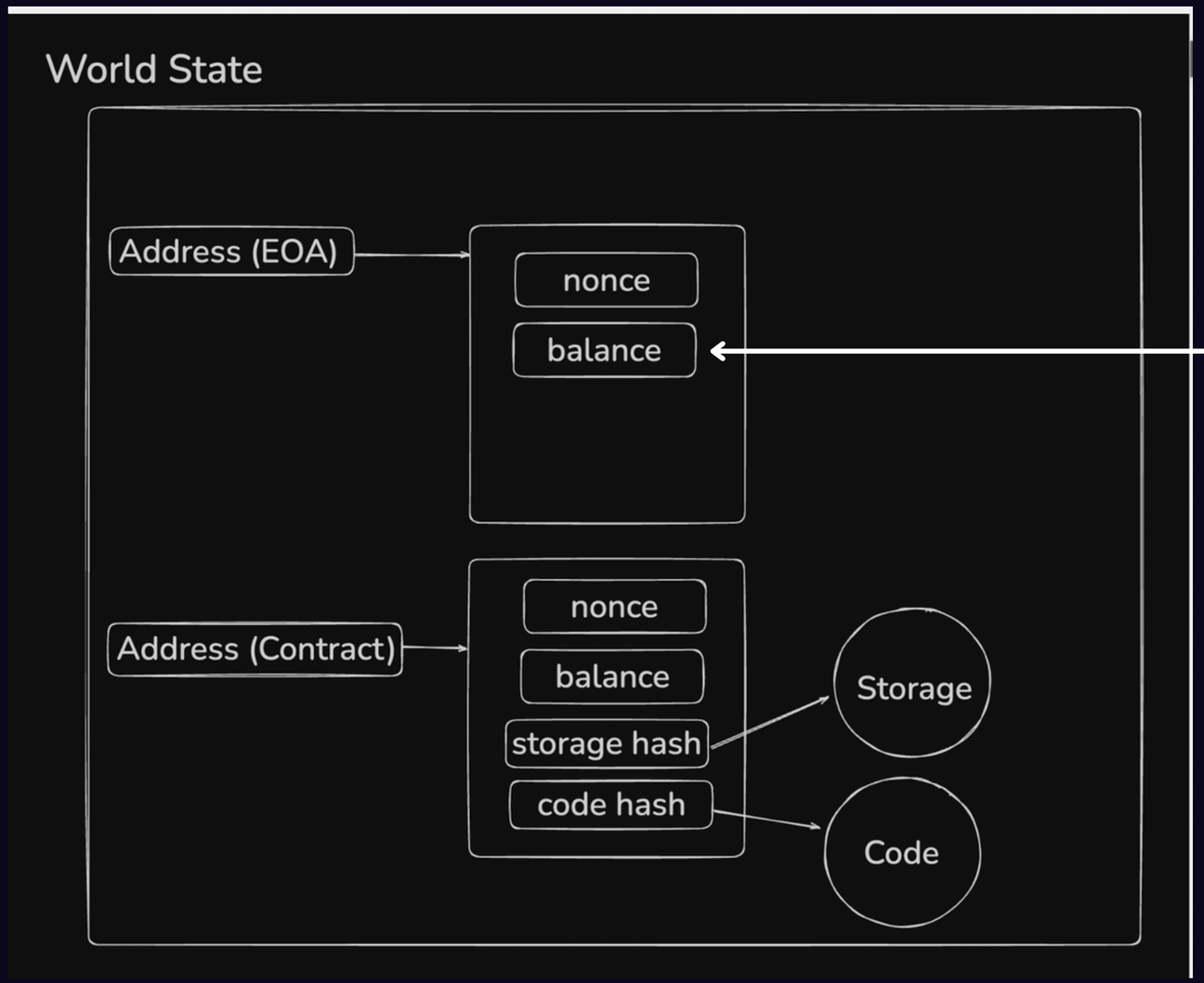
What are custom tokens?

Any token other than native ETH on the eth blockchain is a custom token

Custom token

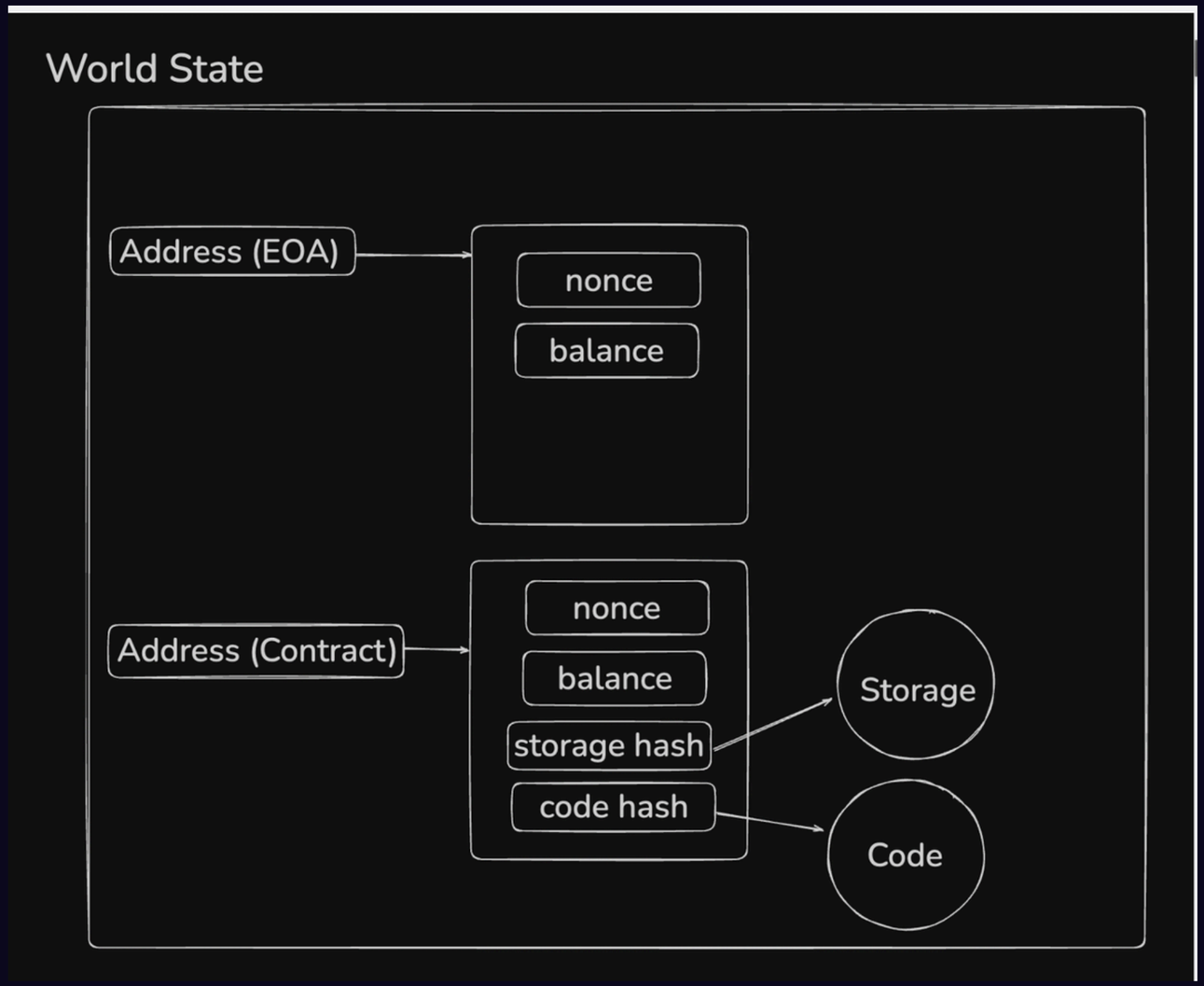


Where are native balances stored?

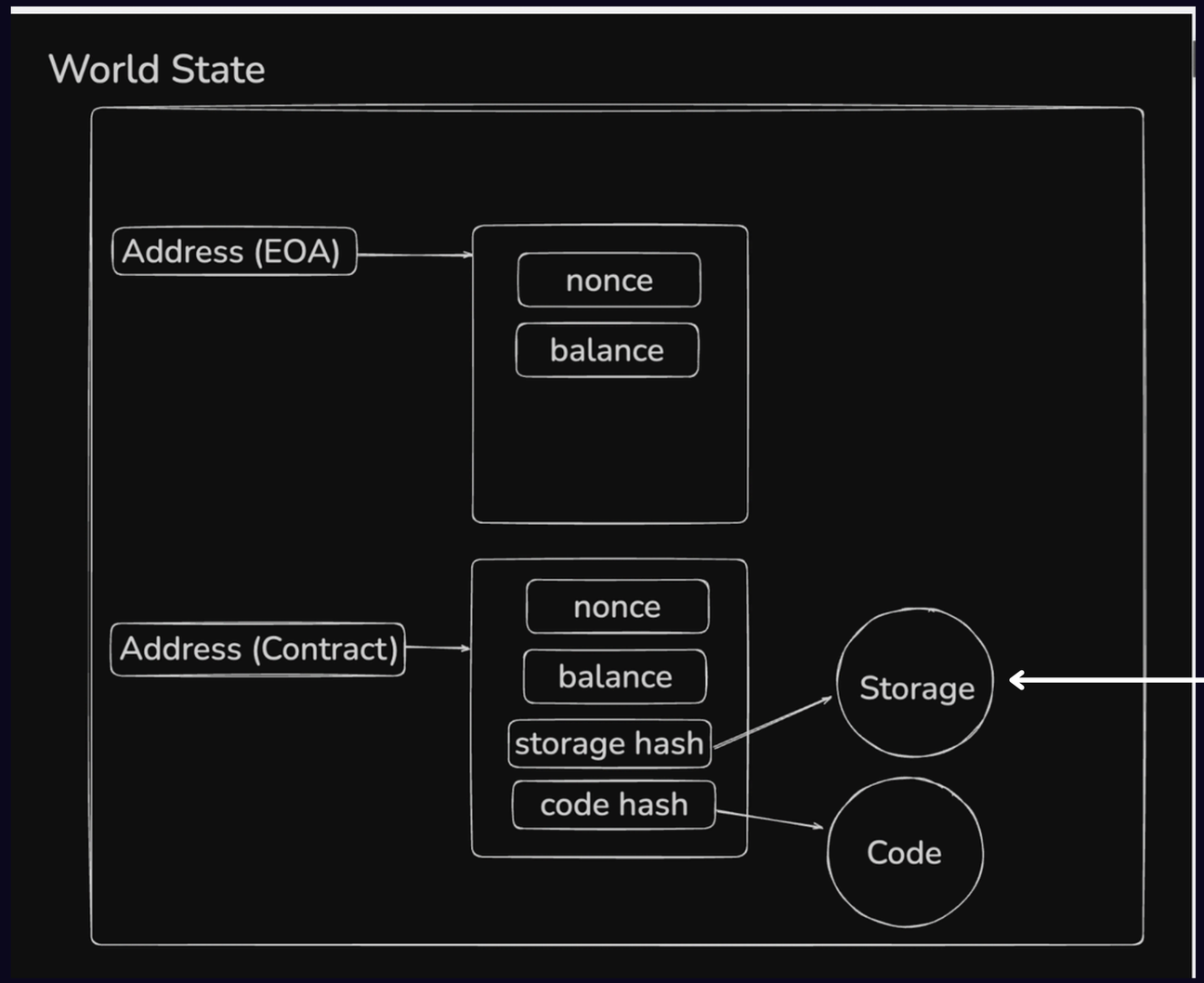


EOA Balance

Where are custom token balances stored?



Where are custom token balances stored?



Custom token balance

Custom token on ETH

Every custom token on ETH is a new contract
This is in contracts to **Solana**, where there is a single Token Program

Custom token on ETH

Every custom token on ETH is a new contract
This is in contracts to **Solana**, where there is a single Token Program

Custom token on ETH

Every custom token on ETH is a new contract
This is in contracts to **Solana**, where there is a single Token Program

USDC on ETH - https://etherscan.io/token/0xa0b86991c6218b36c1d19d4a2e9eb0ce3606eb48
USDC on SOL - https://solscan.io/token/EPjFWdd5AufqSSqeM2qN1xzybapC8G4wEGGkZwyTDt1v

Wtf is a token?

1. It is a type of currency
2. It has an **owner/issuer/minter**
3. There is mapping of which address owns how much of the token
4. Addresses should be allowed to transfer tokens to other addresses

What should a token contract look like?

What should a token contract look like?

1. Initializing the metadata/balances

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Token {
    string public name = "KiratCoin";
    uint public supply = 0;
    address public owner;
    mapping(address => uint) public balances;

    constructor() {  infinite gas 187200 gas
        owner = msg.sender;
    }

}
```

What should a token contract look like?

1. Initializing the metadata/balances
2. Declaring the `mintTo` function

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Token {
    string public name = "KiratCoin";
    uint public supply = 0;
    address public owner;
    mapping(address => uint) public balances;

    constructor() {  infinite gas 280200 gas
        owner = msg.sender;
    }

    function mintTo(address to, uint32 amount) public {
        require(msg.sender == owner);
        balances[to] += amount;
        supply += amount;
    }

}

```

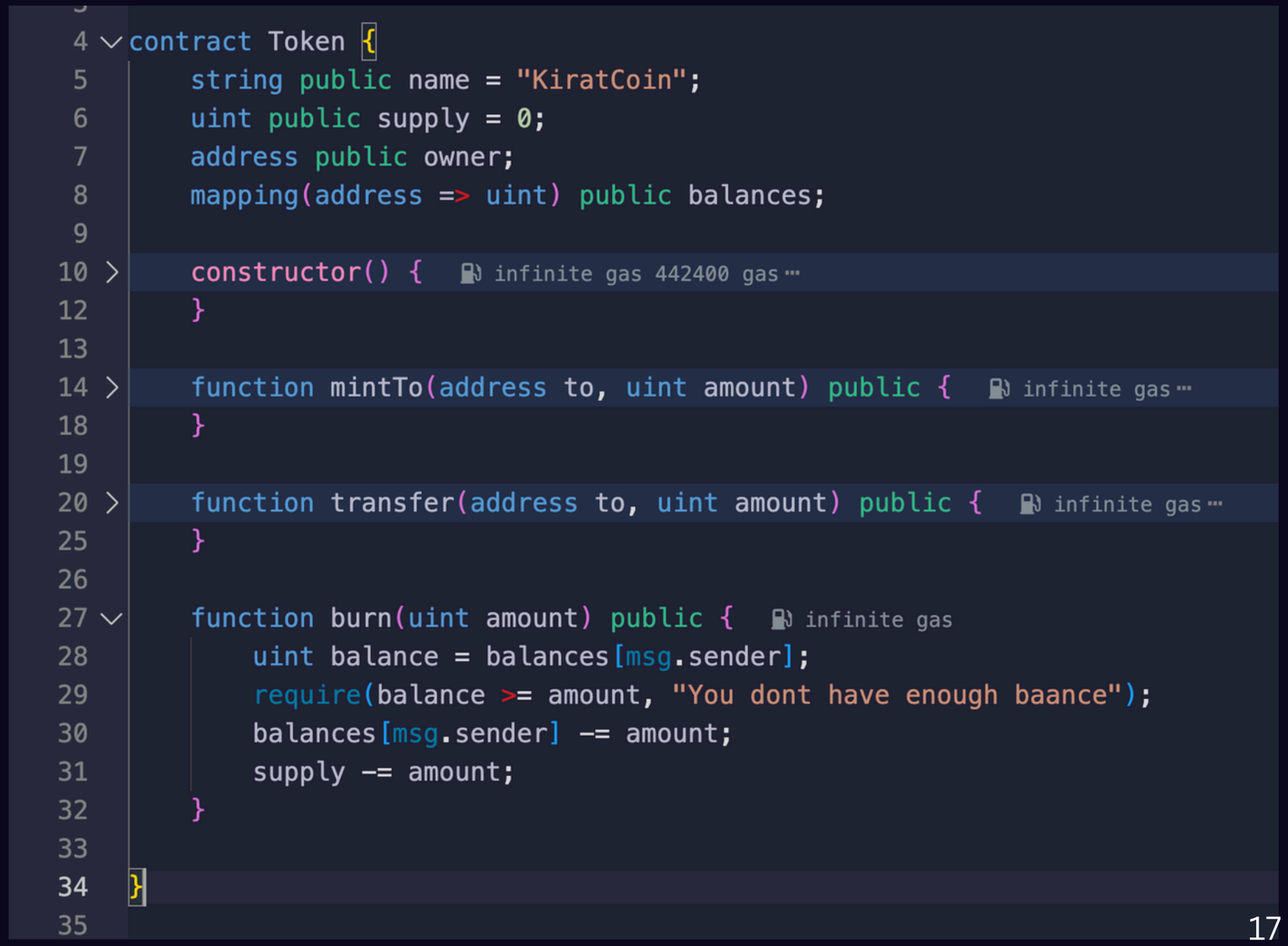
What should a token contract look like?

1. Initializing the metadata/balances
2. Declaring the `mintTo` function
3. Implement the `transfer` function

```
4  contract Token {
5      string public name = "KiratCoin";
6      uint public supply = 0;
7      address public owner;
8      mapping(address => uint) public balances;
9
10     constructor() {   infinite gas 374200 gas
11         owner = msg.sender;
12     }
13
14     function mintTo(address to, uint amount) public {   infinite gas
15         require(msg.sender == owner);
16         balances[to] += amount;
17         supply += amount;
18     }
19
20     function transfer(address to, uint amount) public {   infinite gas
21         uint balance = balances[msg.sender];
22         require(balance >= amount, "You dont have enough baance");
23         balances[msg.sender] -= amount;
24         balances[to] += amount;
25     }
26
```

What should a token contract look like?

1. Initializing the metadata/balances
2. Declaring the `mintTo` function
3. Implement the `transfer` function
4. Implement the `burn` function



Full contract

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Token {
    string public name = "KiratCoin";
    uint public supply = 0;
    address public owner;
    mapping(address => uint) public balances;

    constructor() {
        owner = msg.sender;
    }

    function mintTo(address to, uint amount) public {
        require(msg.sender == owner);
        balances[to] += amount;
        supply += amount;
    }

    function transfer(address to, uint amount) public {
        uint balance = balances[msg.sender];
        require(balance >= amount, "You dont have enough baance");
        balances[msg.sender] -= amount;
        balances[to] += amount;
    }

    function burn(uint amount) public {
        uint balance = balances[msg.sender];
        require(balance >= amount, "You dont have enough baance");
        balances[msg.sender] -= amount;
        supply -= amount;
    }

}
```

https://gist.github.com/hkirat/f05d4ce805e6fd33c88c4b2f171c60cc

Does this contract have everything you need from a token?

Lets read through the ERC-20 spec - https://eips.ethereum.org/EIPS/eip-20

transferFrom

Transfers `_value` amount of tokens from address `_from` to address `_to`, and MUST fire the `Transfer` event.

The `transferFrom` method is used for a withdraw workflow, allowing contracts to transfer tokens on your behalf. This can be used for example to allow a contract to transfer tokens on your behalf and/or to charge fees in sub-currencies. The function SHOULD `throw` unless the `_from` account has deliberately authorized the sender of the message via some mechanism.

Note Transfers of 0 values MUST be treated as normal transfers and fire the `Transfer` event.

```
function transferFrom(address _from, address _to, uint256 _value) public returns (bool success)
```

approve

Allows `_spender` to withdraw from your account multiple times, up to the `_value` amount. If this function is called again it overwrites the current allowance with `_value`.

NOTE: To prevent attack vectors like the one described here and discussed here, clients SHOULD make sure to create user interfaces in such a way that they set the allowance first to `0` before setting it to another value for the same spender. THOUGH The contract itself shouldn't enforce it, to allow backwards compatibility with contracts deployed before

```
function approve(address _spender, uint256 _value) public returns (bool success)
```

allowance

Returns the amount which `_spender` is still allowed to withdraw from `_owner`.

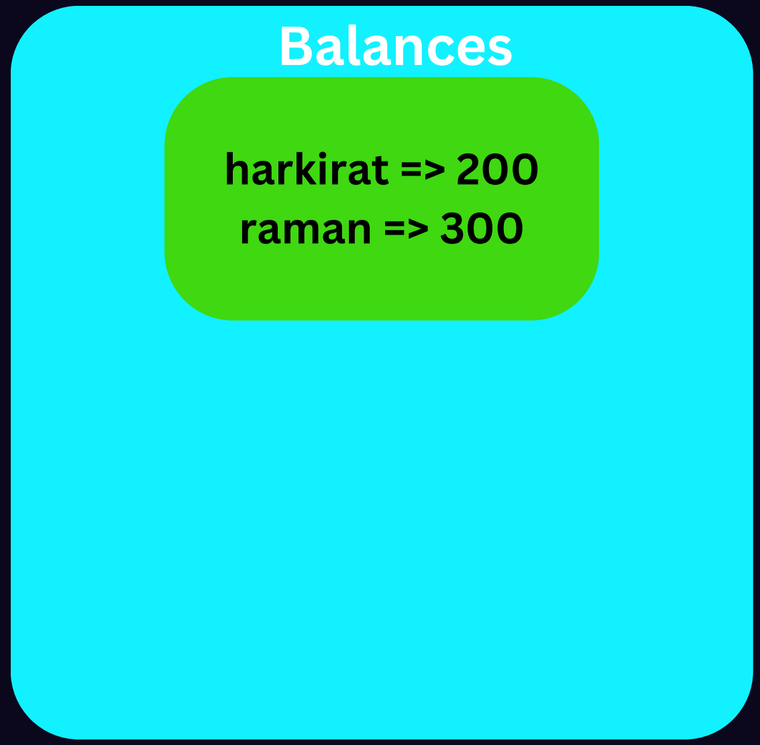
```
function allowance(address _owner, address _spender) public view returns (uint256 remaining)
```

Allowances/Delegation

An allowance refers to the mechanism that allows a token holder to grant permission to another address (typically a smart contract) to spend a specified amount of their tokens.

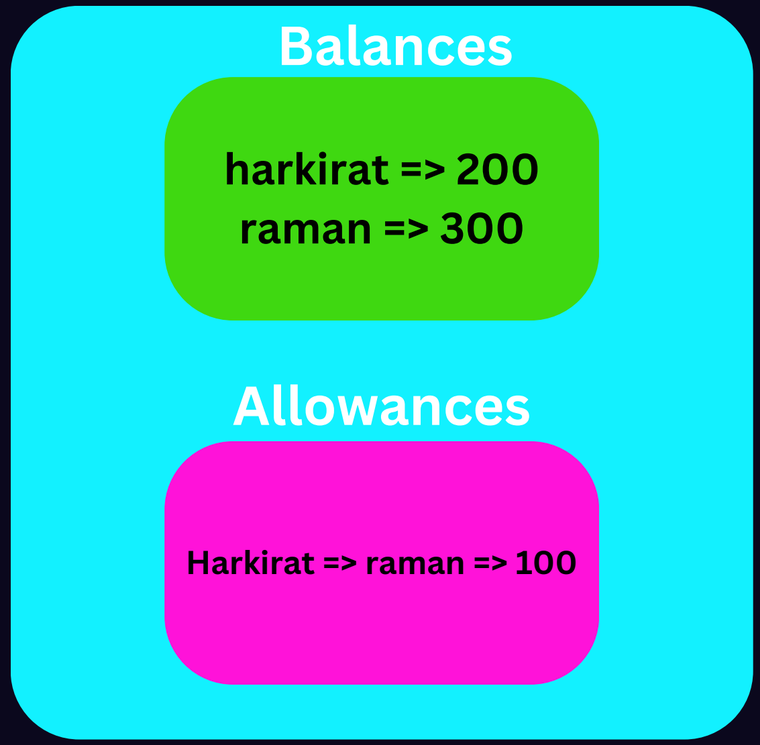
Allowances/Delegation

An allowance refers to the mechanism that allows a token holder to grant permission to another address (typically a smart contract) to spend a specified amount of their tokens.



Allowances/Delegation

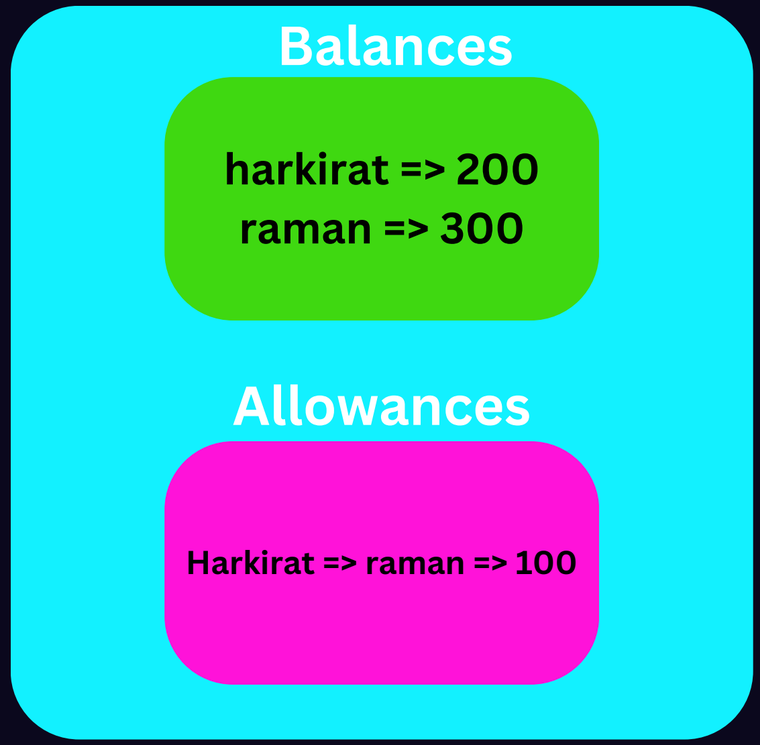
An allowance refers to the mechanism that allows a token holder to grant permission to another address (typically a smart contract) to spend a specified amount of their tokens.



Allowances/Delegation

An allowance refers to the mechanism that allows a token holder to grant permission to another address (typically a smart contract) to spend a specified amount of their tokens.

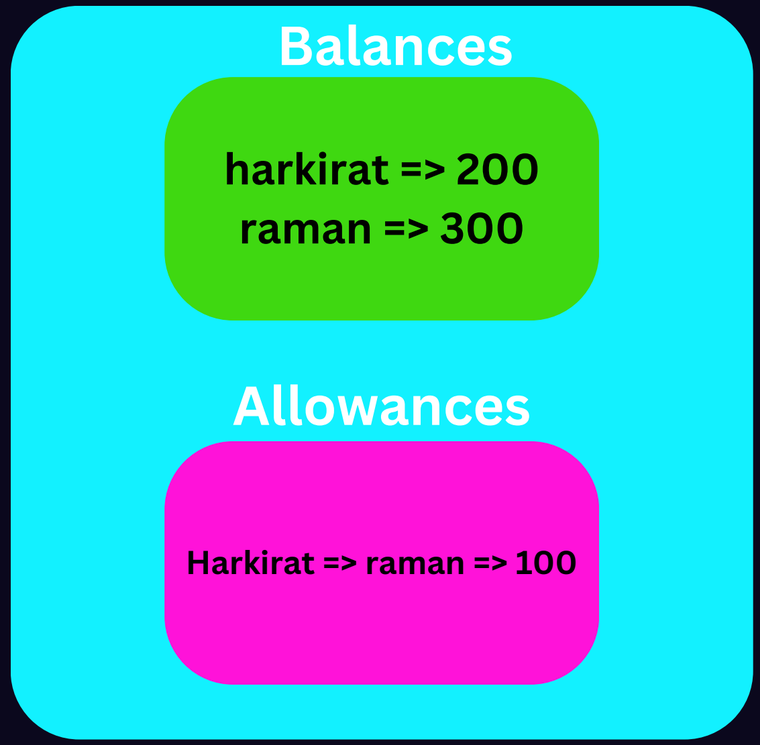
This means raman can spend 100 tokens of harkirat



Allowances/Delegation

An allowance refers to the mechanism that allows a token holder to grant permission to another address (typically a smart contract) to spend a specified amount of their tokens.

This means raman can spend 100 tokens of harkirat



Allowances/Delegation

This means raman can spend 100 tokens of harkirat

Allowances/Delegation

This means raman can spend 100 tokens of harkirat

Allowances/Delegation

This means raman can spend 100 tokens of harkirat

How should the contract change?

```
mapping(address account => mapping(address sender =>  uint)) public allowance;


  function transferFrom(address from, address to, uint amount) public {
      uint balance = balances[from];
      require(balance >= amount);
      uint currentAllowance = allowance[from][msg.sender];
      require(currentAllowance >= amount);

      balances[from] -= amount;
      balances[to] += amount;
      allowance[from][msg.sender] -= amount;
  }
```

Full contract

```sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Token {
    string public name = "KiratCoin";
    uint public supply = 0;
    address public owner;
    mapping(address => uint) public balances;
    mapping(address account => mapping(address sender =>  uint)) public allowance;

    constructor() {
        owner = msg.sender;
    }

    function mintTo(address to, uint amount) public {
        require(msg.sender == owner);
        balances[to] += amount;
        supply += amount;
    }

    function transfer(address to, uint amount) public {
        uint balance = balances[msg.sender];
        require(balance >= amount, "You dont have enough baance");
        balances[msg.sender] -= amount;
        balances[to] += amount;
    }

    function burn(uint amount) public {
        uint balance = balances[msg.sender];
        require(balance >= amount, "You dont have enough baance");
        balances[msg.sender] -= amount;
        supply -= amount;
    }

    function allow(address to, uint amount) public {
        allowance[msg.sender][to] = amount;
    }

    function transferFrom(address from, address to, uint amount) public {
        uint balance = balances[from];
        require(balance >= amount);
        uint currentAllowance = allowance[from][msg.sender];
        require(currentAllowance >= amount);

        balances[from] -= amount;
        balances[to] += amount;
        allowance[from][msg.sender] -= amount;
    }

}
```

https://gist.github.com/hkirat/7ded3b20325c32372b014e2a0013b72b

Is something else missing?

Is something else missing?

Events

Transfer

MUST trigger when tokens are transferred, including zero value transfers.

A token contract which creates new tokens SHOULD trigger a Transfer event with the `_from` address set to `0x0` when tokens are created.

```
event Transfer(address indexed _from, address indexed _to, uint256 _value)
```

Approval

MUST trigger on any successful call to `approve(address _spender, uint256 _value)`.

```
event Approval(address indexed _owner, address indexed _spender, uint256 _value)
```

Is something else missing?

```
event Transfer(address indexed from, address indexed to, uint256 value);
event Approval(address indexed owner, address indexed spender, uint256 value);

function allow(address to, uint amount) public {   infinite gas
    allowance[msg.sender][to] = amount;
    emit Approval(msg.sender, to, amount);
}

function transferFrom(address from, address to, uint amount) public {
    uint balance = balances[from];
    require(balance >= amount);
    uint currentAllowance = allowance[from][msg.sender];
    require(currentAllowance >= amount);

    balances[from] -= amount;
    balances[to] += amount;
    allowance[from][msg.sender] -= amount;
    emit Transfer(from, to, amount);
}
```

Is something else missing?

Is something else missing?

Decimals

name

Returns the name of the token - e.g. `"MyToken"`.

OPTIONAL - This method can be used to improve usability, but interfaces and other contracts MUST NOT expect these values to be present.

```
function name() public view returns (string)
```

symbol

Returns the symbol of the token. E.g. “HIX”.

OPTIONAL - This method can be used to improve usability, but interfaces and other contracts MUST NOT expect these values to be present.

```
function symbol() public view returns (string)
```

decimals

Returns the number of decimals the token uses - e.g. `8`, means to divide the token amount by `100000000` to get its user representation.

OPTIONAL - This method can be used to improve usability, but interfaces and other contracts MUST NOT expect these values to be present.

```
function decimals() public view returns (uint8)
```

totalSupply

Returns the total token supply.

```
function totalSupply() public view returns (uint256)
```

Is something else missing?

Decimals

The decimals parameter refers to the number of decimal places the token can be divided into. It is an important property that defines the smallest unit of the token, enabling more granular transactions and better user experience in terms of handling very small amounts of the token.

ERC-20 spec

totalSupply()
Returns the total number of tokens in circulation (the total supply of the token).

balanceOf(address account)
Returns the balance of tokens held by a specific address (account).

transfer(address recipient, uint256 amount)
Transfers a specified number of tokens (amount) from the caller’s account to the recipient address. This function returns true if the transfer is successful.

allowance(address owner, address spender)
Returns the amount of tokens that the spender is allowed to transfer on behalf of the owner based on the allowance previously set by the owner using the approve() function.

approve(address spender, uint256 amount)
Approves the spender to spend up to a specified number of tokens (amount) from the caller's account. This creates an "allowance" that the spender can use with the transferFrom() function.

transferFrom(address sender, address recipient, uint256 amount)
Allows the spender (who must have been approved by the sender) to transfer tokens from the sender's account to the recipient's account.

ERC-20 spec

Total supply in circulation ⟶

totalSupply()
Returns the total number of tokens in circulation (the total supply of the token).

balanceOf(address account)
Returns the balance of tokens held by a specific address (account).

transfer(address recipient, uint256 amount)
Transfers a specified number of tokens (amount) from the caller's account to the recipient address. This function returns true if the transfer is successful.

allowance(address owner, address spender)
Returns the amount of tokens that the spender is allowed to transfer on behalf of the owner based on the allowance previously set by the owner using the approve() function.

approve(address spender, uint256 amount)
Approves the spender to spend up to a specified number of tokens (amount) from the caller's account. This creates an "allowance" that the spender can use with the transferFrom() function.

transferFrom(address sender, address recipient, uint256 amount)
Allows the spender (who must have been approved by the sender) to transfer tokens from the sender's account to the recipient's account.

ERC-20 spec

balance of a specific address ⟶

totalSupply()
Returns the total number of tokens in circulation (the total supply of the token).

balanceOf(address account)
Returns the balance of tokens held by a specific address (account).

transfer(address recipient, uint256 amount)
Transfers a specified number of tokens (amount) from the caller’s account to the recipient address. This function returns true if the transfer is successful.

allowance(address owner, address spender)
Returns the amount of tokens that the spender is allowed to transfer on behalf of the owner based on the allowance previously set by the owner using the approve() function.

approve(address spender, uint256 amount)
Approves the spender to spend up to a specified number of tokens (amount) from the caller's account. This creates an "allowance" that the spender can use with the transferFrom() function.

transferFrom(address sender, address recipient, uint256 amount)
Allows the spender (who must have been approved by the sender) to transfer tokens from the sender's account to the recipient's account.

ERC-20 spec

Transfer from address A to B

totalSupply()
Returns the total number of tokens in circulation (the total supply of the token).

balanceOf(address account)
Returns the balance of tokens held by a specific address (account).

transfer(address recipient, uint256 amount)
Transfers a specified number of tokens (amount) from the caller's account to the recipient address. This function returns true if the transfer is successful.

allowance(address owner, address spender)
Returns the amount of tokens that the spender is allowed to transfer on behalf of the owner based on the allowance previously set by the owner using the approve() function.

approve(address spender, uint256 amount)
Approves the spender to spend up to a specified number of tokens (amount) from the caller's account. This creates an "allowance" that the spender can use with the transferFrom() function.

transferFrom(address sender, address recipient, uint256 amount)
Allows the spender (who must have been approved by the sender) to transfer tokens from the sender's account to the recipient's account.

ERC-20 spec

Allowance of owner on spender ⟶

totalSupply()
Returns the total number of tokens in circulation (the total supply of the token).

balanceOf(address account)
Returns the balance of tokens held by a specific address (account).

transfer(address recipient, uint256 amount)
Transfers a specified number of tokens (amount) from the caller’s account to the recipient address. This function returns true if the transfer is successful.

allowance(address owner, address spender)
Returns the amount of tokens that the spender is allowed to transfer on behalf of the owner based on the allowance previously set by the owner using the approve() function.

approve(address spender, uint256 amount)
Approves the spender to spend up to a specified number of tokens (amount) from the caller's account. This creates an "allowance" that the spender can use with the transferFrom() function.

transferFrom(address sender, address recipient, uint256 amount)
Allows the spender (who must have been approved by the sender) to transfer tokens from the sender's account to the recipient's account.

ERC-20 spec

totalSupply()
Returns the total number of tokens in circulation (the total supply of the token).

balanceOf(address account)
Returns the balance of tokens held by a specific address (account).

transfer(address recipient, uint256 amount)
Transfers a specified number of tokens (amount) from the caller's account to the recipient address. This function returns true if the transfer is successful.

allowance(address owner, address spender)
Returns the amount of tokens that the spender is allowed to transfer on behalf of the owner based on the allowance previously set by the owner using the approve() function.

Allow spender to spend X ⟶

approve(address spender, uint256 amount)
Approves the spender to spend up to a specified number of tokens (amount) from the caller's account. This creates an "allowance" that the spender can use with the transferFrom() function.

transferFrom(address sender, address recipient, uint256 amount)
Allows the spender (who must have been approved by the sender) to transfer tokens from the sender's account to the recipient's account.

ERC-20 spec

totalSupply()
Returns the total number of tokens in circulation (the total supply of the token).

balanceOf(address account)
Returns the balance of tokens held by a specific address (account).

transfer(address recipient, uint256 amount)
Transfers a specified number of tokens (amount) from the caller’s account to the recipient address. This function returns true if the transfer is successful.

allowance(address owner, address spender)
Returns the amount of tokens that the spender is allowed to transfer on behalf of the owner based on the allowance previously set by the owner using the approve() function.

approve(address spender, uint256 amount)
Approves the spender to spend up to a specified number of tokens (amount) from the caller's account. This creates an "allowance" that the spender can use with the transferFrom() function.

Spender transfers from your acc ⟶

transferFrom(address sender, address recipient, uint256 amount)
Allows the spender (who must have been approved by the sender) to transfer tokens from the sender's account to the recipient's account.

ERC-20 spec

Whats missing?

totalSupply()
Returns the total number of tokens in circulation (the total supply of the token).

balanceOf(address account)
Returns the balance of tokens held by a specific address (account).

transfer(address recipient, uint256 amount)
Transfers a specified number of tokens (amount) from the caller’s account to the recipient address. This function returns true if the transfer is successful.

allowance(address owner, address spender)
Returns the amount of tokens that the spender is allowed to transfer on behalf of the owner based on the allowance previously set by the owner using the approve() function.

approve(address spender, uint256 amount)
Approves the spender to spend up to a specified number of tokens (amount) from the caller's account. This creates an "allowance" that the spender can use with the transferFrom() function.

transferFrom(address sender, address recipient, uint256 amount)
Allows the spender (who must have been approved by the sender) to transfer tokens from the sender's account to the recipient's account.

ERC-20 spec

Whats missing?

Minting and burning logic

The ERC-20 standard only defines the basic functions and events that a token contract must implement to ensure interoperability across platforms (wallets, exchanges, etc.), such as transferring tokens, checking balances, and setting allowances.

totalSupply()
Returns the total number of tokens in circulation (the total supply of the token).

balanceOf(address account)
Returns the balance of tokens held by a specific address (account).

transfer(address recipient, uint256 amount)
Transfers a specified number of tokens (amount) from the caller’s account to the recipient address. This function returns true if the transfer is successful.

allowance(address owner, address spender)
Returns the amount of tokens that the spender is allowed to transfer on behalf of the owner based on the allowance previously set by the owner using the approve() function.

approve(address spender, uint256 amount)
Approves the spender to spend up to a specified number of tokens (amount) from the caller's account. This creates an "allowance" that the spender can use with the transferFrom() function.

transferFrom(address sender, address recipient, uint256 amount)
Allows the spender (who must have been approved by the sender) to transfer tokens from the sender's account to the recipient's account.

How to mint/launch a token?

1. Initial Coin Offering (ICO)
2. Airdrop
3. Fair Launch
4. Token Vesting / Token Unlocking

Common implemnetations of ERC-20

Implementation

There are already plenty of ERC20-compliant tokens deployed on the Ethereum network. Different implementations have been written by various teams that have different trade-offs: from gas saving to improved security.

Example implementations are available at

- OpenZeppelin implementation
- ConsenSys implementation

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol

```solidity
/**
 * @dev Transfers a `value` amount of tokens from `from` to `to`, or alternatively mints (or burns) if `from`
 * (or `to`) is the zero address. All customizations to transfers, mints, and burns should be done by overriding
 * this function.
 *
 * Emits a {Transfer} event.
 */
function _update(address from, address to, uint256 value) internal virtual {
    if (from == address(0)) {
        // Overflow check required: The rest of the code assumes that totalSupply never overflows
        _totalSupply += value;
    } else {
        uint256 fromBalance = _balances[from];
        if (fromBalance < value) {
            revert ERC20InsufficientBalance(from, fromBalance, value);
        }
        unchecked {
            // Overflow not possible: value <= fromBalance <= totalSupply.
            _balances[from] = fromBalance - value;
        }
    }

    if (to == address(0)) {
        unchecked {
            // Overflow not possible: value <= totalSupply or value <= fromBalance <= totalSupply.
            _totalSupply -= value;
        }
    } else {
        unchecked {
            // Overflow not possible: balance + value is at most totalSupply, which we know fits into a uint256.
            _balances[to] += value;
        }
    }

    emit Transfer(from, to, value);
}
```

```solidity
/**
 * @dev Transfers a `value` amount of tokens from `from` to `to`, or alternatively mints (or burns) if `from`
 * (or `to`) is the zero address. All customizations to transfers, mints, and burns should be done by overriding
 * this function.
 *
 * Emits a {Transfer} event.
 */
function _update(address from, address to, uint256 value) internal virtual {
    if (from == address(0)) {
        // Overflow check required: The rest of the code assumes that totalSupply never overflows
        _totalSupply += value;
    } else {
        uint256 fromBalance = _balances[from];
        if (fromBalance < value) {
            revert ERC20InsufficientBalance(from, fromBalance, value);
        }
        unchecked {
            // Overflow not possible: value <= fromBalance <= totalSupply.
            _balances[from] = fromBalance - value;
        }
    }

    if (to == address(0)) {
        unchecked {
            // Overflow not possible: value <= totalSupply or value <= fromBalance <= totalSupply.
            _totalSupply -= value;
        }
    } else {
        unchecked {
            // Overflow not possible: balance + value is at most totalSupply, which we know fits into a uint256.
            _balances[to] += value;
        }
    }

    emit Transfer(from, to, value);
}
```

minting

```
/**
 * @dev Transfers a `value` amount of tokens from `from` to `to`, or alternatively mints (or burns) if `from`
 * (or `to`) is the zero address. All customizations to transfers, mints, and burns should be done by overriding
 * this function.
 *
 * Emits a {Transfer} event.
 */
function _update(address from, address to, uint256 value) internal virtual {
    if (from == address(0)) {
        // Overflow check required: The rest of the code assumes that totalSupply never overflows
        _totalSupply += value;
    } else {
        uint256 fromBalance = _balances[from];
        if (fromBalance < value) {
            revert ERC20InsufficientBalance(from, fromBalance, value);
        }
        unchecked {
            // Overflow not possible: value <= fromBalance <= totalSupply.
            _balances[from] = fromBalance - value;
        }
    }

    if (to == address(0)) {
        unchecked {
            // Overflow not possible: value <= totalSupply or value <= fromBalance <= totalSupply.
            _totalSupply -= value;
        }
    } else {
        unchecked {
            // Overflow not possible: balance + value is at most totalSupply, which we know fits into a uint256.
            _balances[to] += value;
        }
    }

    emit Transfer(from, to, value);
}
```

burning →

```
/**
 * @dev Transfers a `value` amount of tokens from `from` to `to`, or alternatively mints (or burns) if `from`
 * (or `to`) is the zero address. All customizations to transfers, mints, and burns should be done by overriding
 * this function.
 *
 * Emits a {Transfer} event.
 */
function _update(address from, address to, uint256 value) internal virtual {
    if (from == address(0)) {
        // Overflow check required: The rest of the code assumes that totalSupply never overflows
        _totalSupply += value;
    } else {
        uint256 fromBalance = _balances[from];
        if (fromBalance < value) {
            revert ERC20InsufficientBalance(from, fromBalance, value);
        }
        unchecked {
            // Overflow not possible: value <= fromBalance <= totalSupply.
            _balances[from] = fromBalance - value;
        }
    }

    if (to == address(0)) {
        unchecked {
            // Overflow not possible: value <= totalSupply or value <= fromBalance <= totalSupply.
            _totalSupply -= value;
        }
    } else {
        unchecked {
            // Overflow not possible: balance + value is at most totalSupply, which we know fits into a uint256.
            _balances[to] += value;
        }
    }

    emit Transfer(from, to, value);
}
```

Create your own token

Write your own implementation of ERC-20

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Token {
    string public name = "KiratCoin";
    uint public supply = 0;
    address public owner;
    mapping(address => uint) public balances;
    mapping(address account => mapping(address sender =>  uint)) public allowance;

    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);

    constructor() {
        owner = msg.sender;
    }

    function mintTo(address to, uint amount) public {
        require(msg.sender == owner);
        balances[to] += amount;
        supply += amount;
    }

    function transfer(address to, uint amount) public {
        uint balance = balances[msg.sender];
        require(balance >= amount, "You dont have enough baance");
        balances[msg.sender] -= amount;
        balances[to] += amount;
        emit Transfer(msg.sender, to, amount);
    }

    function burn(uint amount) public {
        uint balance = balances[msg.sender];
        require(balance >= amount, "You dont have enough baance");
        balances[msg.sender] -= amount;
        supply -= amount;
    }

    function allow(address to, uint amount) public {
        allowance[msg.sender][to] = amount;
        emit Approval(msg.sender, to, amount);
    }

    function transferFrom(address from, address to, uint amount) public {
        uint balance = balances[from];
        require(balance >= amount);
        uint currentAllowance = allowance[from][msg.sender];
        require(currentAllowance >= amount);

        balances[from] -= amount;
        balances[to] += amount;
        allowance[from][msg.sender] -= amount;
        emit Transfer(from, to, amount);
    }

}
```

Extend an audited contract

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

contract MyMintableToken is ERC20, Ownable {

    constructor() ERC20("MyMintableToken", "MMT") Ownable(msg.sender) {
        _mint(msg.sender, 1000000000);
    }

    function mint(address account, uint256 amount) public onlyOwner {
        _mint(account, amount);
    }
}
```

https://gist.github.com/hkirat/b6e5bfa51c878c29c84b9d3964243703