Title	ERC-20 EVENT AND FUNCTION + SAMPLE SMART CONTRACT
Description	Intro
Date	June 18th, 2022
Author	Arashtad
Author URI	https://Arashtad.com

In this article, we are going to analyze the ERC-20 event and function deeper and get familiar with the building blocks of any ERC-20 smart contract. Learning these methods and events will help us write the smart contract of our desired token. Throughout this article, you will see the functionalities of all the smart contracts that we wrote for creating an ERC20 token in the last article.

FUNCTIONS AND EVENT OF ALL ERC-20 CONTRACTS

In general, every ERC-20 token has the below functions and events:

```
function name() public view returns (string)
function symbol() public view returns (string)
function decimals() public view returns (uint8)
function totalSupply() public view returns (uint256)
function balanceOf(address _owner) public view returns (uint256
balance)
function transfer(address _to, uint256 _value) public returns (bool
success)
function transferFrom(address _from, address _to, uint256 _value)
public returns (bool success)
function approve(address _spender, uint256 _value) public returns
(bool success)
function allowance(address _owner, address _spender) public view
returns (uint256 remaining)
```

ERC-20 EVENT

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

A SAMPLE ERC-20 SMART CONTRACT: METHOD & EVENT

For the following contract, we are going to explain more about the ERC-20 method and event (TokenERC20.sol):

```
pragma solidity ^0.6.0;

interface tokenRecipient {
     function receiveApproval(address _from, uint256 _value, address
_token, bytes calldata
                _extraData) external;
}

contract TokenERC20 {
     string public name;
     string public symbol;
     uint8 public decimals = 18;
     uint256 public totalSupply;
}
```

Name, symbol, decimals, and total supply are the main variables of every ERC-20 smart contract.

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

Created a mapping between the addresses and balances (array of addresses and balances).

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

A public event that will inform the clients about the transfer (sender, receiver, and value sent).

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

A public event that will inform the clients about the transfer (sender, receiver, and value sent).

```
event Burn(address indexed from, uint256 value);
```

The burn event will inform the clients about the amount that has been burnt.

```
constructor(uint256 initialSupply,
          string memory tokenName,
          string memory tokenSymbol
          ) public {
     totalSupply = initialSupply * 10 ** uint256(decimals);
     balanceOf[msg.sender] = totalSupply;
     name = tokenName;
     symbol = tokenSymbol;
}
```

The above constructor determines the total supply according to the decimals. And transfers all of it to the balance of the owner of the contract. In end, the name and the symbol of the token are determined for display.

```
function _transfer(address _from, address _to, uint _value) internal
{
     require(_to != address(0x0));
}
```

The receiver should not be addressed as 0x0 or the owner of the contract.

```
require(balanceOf[_from] >= _value);
```

The sender must have enough balance to send the desired value.

```
require(balanceOf[_to] + _value >= balanceOf[_to]);
```

After the transaction, the receiver's balance must increase. In other words, the value must be greater or equal to zero.

```
uint previousBalances = balanceOf[_from] + balanceOf[_to];
```

Before the transfer, the balance of the sender and receiver is calculated for the later tests.

```
balanceOf[_from] -= _value;
```

After the transfer, the balance of the sender is subtracted by the value.

```
balanceOf[_to] += _value;
```

After the transfer, the balance of the receiver is added to the value of the transfer.

```
emit Transfer(_from, _to, _value);
```

The event is emitted to notify the clients about the transaction.

```
assert(balanceOf[_from] + balanceOf[_to] == previousBalances);
```

We test whether, after the transfer, the sum of the balances equals the previous sum of balances.The following function uses the data of the _transfer to verify the success of the transaction.

```
function transfer(address _to, uint256 _value) public returns (bool
success) {
    _transfer(msg.sender, _to, _value);
    return true;
}
```

The following function will allow the sender to send just the value to the receiver.

```
function transferFrom(address _from, address _to, uint256 _value)
public returns (bool success) {
    require(_value <= allowance[_from][msg.sender]);
    allowance[_from][msg.sender] -= _value;
    _transfer(_from, _to, _value);
    return true;
}
```

The function below checks the output of the above function if it has returned true and will let the contract know.

```
function approveAndCall(address _spender, uint256 _value, bytes
memory
     _extraData) public returns (bool success) {
     tokenRecipient spender = tokenRecipient(_spender);
     if (approve(_spender, _value)) {
          spender.receiveApproval(msg.sender, _value, address(this),
_extraData);
          return true;
     }
}
```

The function below burns the amount of transfer value, or in other words, it will decrease the amount that has been transferred from the total supply.

```
function burn(uint256 _value) public returns (bool success) {
     require(balanceOf[msg.sender] >= _value);
     balanceOf[msg.sender] -= _value; // Subtract from the sender
     totalSupply -= _value;
     emit Burn(msg.sender, _value);
     return true;
}
```

The following function is the same as above with the difference that it will ask for the allowance of the sender for the burning.

```
function burnFrom(address _from, uint256 _value) public returns
(bool success) {
     require(balanceOf[_from] >= _value);
     require(_value <= allowance[_from][msg.sender]);
     balanceOf[_from] -= _value
     allowance[_from][msg.sender] -= _value;

     totalSupply -= _value;
     emit Burn(_from, _value);
     return true;
}
```

FINAL WORD

In this article, we have studied an ERC-20 smart contract. We have analyzed the necessary functions, methods, and events that every ERC-20 token smart contract must have. Furthermore, we have taken a look at how a sample token contract has been written in solidity with the consideration of all the events, methods, and necessary functions. Notice, that you can write an ERC-20 smart contract in a very different way as long as you include the necessary method and event in it.