Decentralized Tokens Swap System

The goal of this tutorial is to describe the inner working of a decentralized Tokens Swap System.
Let's briefly explain some terms before deep diving into the topic.

Brief description of the environment

A token is like a crypto-currency inside a blockchain. It is written as a smart-contract. A smart-contract is no more than a program saved at a certain address in the blockchain and executed on it.
On a blockchain, we can find 2 main types of tokens : fungible tokens and non-fungible tokens.
Let's explain what the difference is between them.

When a token is equivalent in value with another token of the same type, they are said to be fungible. For example, a person A has borrowed 10 Euros from a person B. A gave a note of 10 Euros to B. Later, B reimburse A with another note of 10 Euros. Of course, B has not to reimburse A with the same note that A gave to him. Any 10 Euros note, 2 notes of 5 Euros or anything else which sum is 10 Euros is acceptable by A. 1 Euro is equivalent to another 1 Euro.That's what is fungibility. Euro is said to be fungible.

In Ethereum environment, this type of token is called ERC-20.

On the contrary, when a token is not equivalent in value with another token, they are said non-fungible. For example, a specific artwork has not necessarily the same value as another artwork. Every artwork is unique.

In Ethereum environment, this type of token, NFT, is called ERC-721.

This tutorial, in the following, is oriented to fungible tokens.

A token is composed of several features, here is the main ones :

- a name
- a symbol
- a number of decimals
- a total supply, equivalent of the monetary mass of the token
- a function that returns the balance of this token of a specific address
- a function that transfers a number of tokens from address A to address B.

The complete specification is described in the Technical Working section.

Inner working of a Token Swap System

What is a Token Swap System

The goal of aToken Swap System is to swap a certain amount of tokens X to a certain amount of tokens Y between 2 addresses. Let's see it as an example. Let's consider that address A has an amount of x tokens of type ERC20token1 and that address B has an amount of y tokens of type ERC20token2. Both ERC20token1 and ERC20token2 are fungible tokens.

Before swap :

	Address A	Address B
Balance of ERC20token1	a1	b1

Balance of ERC20token2	a2	b2

After swap :

	Address A	Address B
Balance of ERC20token1	a1-x	b1+x
Balance of ERC20token2	a2+y	b2-y

Technical working

Fungible Token Specification

Here is the complete specification of a fungible token :

- name(): return the name of the token
- symbol(): return the symbol of the token
- totalSupply(): return the number of tokens initially created
- decimals(): return the maximum number of digits being part of the fractional part of the token
- balanceOf(addr: Address): return the balance of tokens of a specific address
- transferFrom(fromAddress: Address, toAddress: Address, amount: u64): transfers amount of tokens from fromAddress to toAddress
- transfer(toAddress: Address, amount: u64): transfers amount of tokens to toAddress
- a transfer event
- an approval event

All these functionalities have to be implemented when creating a fungible token.

Transfer transaction mechanism

To see how the swap mechanism works, let's see how a transfer transaction works before.

Let's assume that A wants to send x tokens of type X to B. X can be an instance of ERC20token1 class or ERC20token2 class for example.

A owns the address addressA and B the address addressB.

So, A have to call

```
        ft.transferFrom(addressA, addressB, amount);
```

if ft is an instance of contract of type X.

The function, before making the transfer, has to check how many tokens addressA is authorized to send by the contract caller. This is done by

```
        const owner = Context.caller();

        const spenderAddress: Address = fromAddress;

        const recipient: Address = toAddress;

        const spenderAllowance = this.allowance(owner,
spenderAddress);

        assert(spenderAllowance >= amount,'transferFrom failed:
insufficient allowance',);
```

If it's so, the transfer is made by

```
        assert(this._transfer(spenderAddress, recipient,
amount),'transferFrom failed: invalid amount',);
```

This subtracts amount tokens of X from the balance of addressA and add amount tokens of X to the balance of addressB.

Then, the number of tokens that addressA is authorized to send by the contract caller, in other words its allowance, has to be modified. The allowance must be decreased of amount. This is done by

```
     this.approve(owner, spenderAddress, spenderAllowance -
amount);
```

Note that an approval event is generated on the blockchain when

```
approve(owner: Address, spenderAddress: Address, amount: u64):
void
```

is called. This is done by

```
     generateEvent(

  createEvent(APPROVAL_EVENT_NAME, [

   owner.toString(),

   spenderAddress.toString(),

   amount.toString(),

  ]),

 );
```

And finally, the transfer event has to be generated on the blockchain by

```
     generateEvent(

  createEvent(TRANSFER_EVENT_NAME, [

   fromAddress.toString(),

   toAddress.toString(),

   amount.toString(),
  ]),
```

```
 );
```

Swap mechanism

So, to simulate a Swap System, there are 2 classes that each represent a fungible token: ERC20token1 class and ERC20token2 class. Each class has all required elements of the fungible token specification.

The SwapToken class owns an instance of ERC20token1 and an instance of ERC20token2.

Finally, the swap simply consists in executing :

a transfer of amount1 tokens of type ERC20token1 from addressA to addressB

and

a transfer of amount2 tokens of type ERC20token2 from addressB to addressA.

This is done by

```
     swap(address1: Address, amount1: u64, address2: Address,
amount2: u64): void
     {
     this.eRC20Token1.transferFrom(addressA, addressB, amount1);
     this.eRC20Token2.transferFrom(addressB, addressA, amount2);
     }.
```