1. MiniMe 토큰 컨트랙트 구현 설명(특히 transfer 함수 내부 구현 관련)

1) MiniMe 토큰 설명

`MiniMeToken` 컨트랙트는 특별한 기능을 가진 `ERC20` 토큰이다. `MiniMe` 토큰은 누구든지 기존에 생성된 토큰을 복제할 수 있는 기능을 갖고 있다. 특히 토큰을 복제할 때 ***오리지날 토큰의 특정 시점 블록에의 토큰의 밸런스 분배 상태를 동일하게 복제할 수 있다.*** 따라서 기존의 상태들을 그대로 유지하면서 토큰에 새로운 기능을 추가할 수 있다는 장점이 있다. 그리고 새로 복제된 토큰은 완전히 독립적인 토큰으로서 오리지날 토큰의 상태를 유지하면서 새로운 기능과 함께 동작하게 된다.

(1) Token Controller

`토큰 컨트롤러` 는 자신이 복제한 토큰에 대해서 `generate/destroy/transfer` 를 수행할 수 있다. 토큰 컨트롤러는 일반적인 `EOA` 도 가능하지만 토큰 컨트롤에 대한 규칙을 갖고 있는 별도의 컨트랙트가 될 수 있다. default로 컨트랙트를 복제한 `msg.sender` 가 토큰 컨트롤러가 된다.

`토큰 컨트롤러` 는 `선택사항(optional)` 이며, 만약 사용하지 않을 경우 토큰 컨트롤러를 `0x0` 으로 설정하여 `disable` 할 수 있다.

토큰 컨트롤러는 토큰의 전송을 `freeze` 할 수 있는데, 다음과 같이 `transfersEnabled` 상태를 `false` 로 변경하면 모든 토큰 전송이 금지된다.

```
transfersEnabled = false;
```

2) MiniMe 토큰 컨트랙트

(1) TokenController 컨트랙트

`TokenController` 컨트랙트는 3가지 함수(`proxyPayment()` , `onTransfer()` , `onApprove()`)를 구현해야 한다. 각 3개의 함수는 `MiniMeToken` 에서 어떤 특정 함수가 호출되었을 때, 만약 `controller` 컨트랙트가 등록되어 있는 경우 호출되는 콜백함수라고 볼 수 있다.

`proxyPayment()` 함수는 `MiniMeToken` 에서 `fallback` 함수가 호출되었을 때 전송된 ether를 처리하는 역할을 한다.

```
function proxyPayment(address _owner) public payable returns(bool);
```

`onTransfer()` 함수는 `MiniMeToken` 에서 `transfer()` 또는 `transferFrom()` 함수가 호출되었을 때 필요한 액션을 처리하는 역할을 한다.

```
function onTransfer(address _from, address _to, uint _amount) public returns(bool);
```

`onTransfer()` 함수는 `MiniMeToken` 에서 `approve()` 함수가 호출되었을 때 필요한 액션을 처리하는 역할을 한다.

```
function onApprove(address _owner, address _spender, uint _amount) public
returns(bool);
```

(2) MiniMeTokenFactory 컨트랙트

`MiniMeTokenFactory` 컨트랙트는 새로운 컨트랙트를 복제할 때 사용되는 컨트랙트이다. 이 때 `createCloneToken()` 함수를 사용하여 오리지날 토큰을 복제한다.

그리고 새로운 토큰을 생성할 때 기존 오리지날 토큰의 주소(`_parentToken`)와 어느 블록넘버 시점의 토큰을 복제할 것이지 스냅샷블록(`_snapshotBlock`)을 지정해주어야 한다.

```
function createCloneToken(
  address _parentToken,
  uint _snapshotBlock,
  string _tokenName,
  uint8 _decimalUnits,
  string _tokenSymbol,
  bool _transfersEnabled
) public returns (MiniMeToken) {
  MiniMeToken newToken = new MiniMeToken(
    this,
    _parentToken,
    _snapshotBlock,
    _tokenName,
    _decimalUnits,
    _tokenSymbol,
    _transfersEnabled
    );

  newToken.changeController(msg.sender);
  return newToken;
}
```

`MiniMeToken` 컨트랙트를 deploy하기 전에 `MiniMeTokenFactory` 컨트랙트를 먼저 deploy한다. 그리고 `MiniMeToken` 컨트랙트를 deploy하면서 생성된 factory 컨트랙트 주소를 함께 설정한다. 그리고 나중에 토큰을 복제할 경우 이 factory 컨트랙트의 `createCloneToken()` 함수를 호출하여 토큰을 복제하게 된다.

(3) MiniMeToken 컨트랙트

Checkpoint 타입

`MiniMeToken` 은 `Checkpoint` 구조체를 사용하여 특정 블록 넘버에 저장되었던 value를 기록한다.

```
struct  Checkpoint {
    // `fromBlock` is the block number that the value was generated from
    uint128 fromBlock;

    // `value` is the amount of tokens at a specific block number
    uint128 value;
}
```

다음과 같이 각 주소에 대한 밸런스를 기록할 때 `Checkpoint` 타입을 사용하여 어떤 사용자의 밸런스에 대한 변화가 발생할 때는 해당 ***블록 넘버와 밸런스 value를 함께 기록한다.***

```
mapping (address => Checkpoint[]) balances;
```

Constructor 함수

`MiniMeToken`을 생성할 때는 미리 생성한 `MiniMeTokenFactory` 컨트랙트의 주소(`_tokenFactory`)와 자신의 부모 MiniMeToken 컨트랙트의 주소(`_parentToken`), 그리고 부모 MiniMeToken 컨트랙트의 스냅샷 블록넘버 (`_parentSnapShotBlock`)를 전달해야 한다.

```
function MiniMeToken(
  address _tokenFactory,
  address _parentToken,
  uint _parentSnapShotBlock,
  string _tokenName,
  uint8 _decimalUnits,
  string _tokenSymbol,
  bool _transfersEnabled
) public {
  tokenFactory = MiniMeTokenFactory(_tokenFactory);
  name = _tokenName;                                 // Set the name
  decimals = _decimalUnits;                          // Set the decimals
  symbol = _tokenSymbol;                             // Set the symbol
  parentToken = MiniMeToken(_parentToken);
  parentSnapShotBlock = _parentSnapShotBlock;
  transfersEnabled = _transfersEnabled;
  creationBlock = block.number;
}
```

> ***참고사항 :***
> *만약 부모 컨트랙트가 없는 최초의 토큰일 경우에는 `_parentToken` 값은 `0x00` 을 넣고, `_parentSnapShotBlock` 값은 `0` 을 넣는다.*

createCloneToken() 함수

`createCloneToken()` 함수는 새로운 토큰을 복제할 때 사용하는 함수이며, 복제하고자 하는 시점의 블록넘버 (`_snapshotBlock`)를 함께 전달한다.

그리고 초기에 설정된 `factory` 컨트랙트의 `createCloneToken()` 함수를 호출하여 새로운 `MiniMeToken` 컨트랙트를 생성한다. 이 때 `controller`는 `msg.sender`가 된다.

```
string _cloneTokenName,
uint8 _cloneDecimalUnits,
string _cloneTokenSymbol,
uint _snapshotBlock,
bool _transfersEnabled
) public returns(address) {
  if (_snapshotBlock == 0) _snapshotBlock = block.number;
  MiniMeToken cloneToken = tokenFactory.createCloneToken(
```

```
    this,
    _snapshotBlock,
    _cloneTokenName,
    _cloneDecimalUnits,
    _cloneTokenSymbol,
    _transfersEnabled
    );

  cloneToken.changeController(msg.sender);

  // An event to make the token easy to find on the blockchain
  NewCloneToken(address(cloneToken), _snapshotBlock);
  return address(cloneToken);
}
```

doTransfer() 함수

사용자들이 ERC20 표준 함수인 `transfer()` 함수를 호출하면 `MiniMeToken` 에서는 내부적으로 `doTransfer()` 함수를 사용하여 토큰을 전송한다.

토큰을 전송하려는 주소의 밸런스를 읽어올 때 `balanceOfAt()` 함수를 사용하는데, 이 함수를 사용하면 최근 토큰 밸런스가 현재 컨트랙트가 아닌 부모 컨트랙트에 있을 경우 내부적으로 `getValueAt()` 함수를 `recursive` 하게 호출하여 가장 최근 기록이 저장된 부모 컨트랙트를 추적하여 최종 밸런스를 읽어온다.

```
var previousBalanceFrom = balanceOfAt(_from, block.number);
```

만약 `controller` 컨트랙트가 설정되어 있을 경우 `controller` 컨트랙트의 `onTransfer()` 콜백함수를 호출한다.

```
// Alerts the token controller of the transfer
if (isContract(controller)) {
  require(TokenController(controller).onTransfer(_from, _to, _amount));
}
```

그 다음 `updateValueAtNow()` 함수를 이용하여 `sender` 의 밸런스를 감소시키기고, `receiver` 의 밸런스를 증가시킨다. 이 때 `updateValueAtNow()` 함수는 현재의 블록넘버와 value를 `Checkpoint` 로 기록한다.

```
// First update the balance array with the new value for the address
//  sending the tokens
updateValueAtNow(balances[_from], previousBalanceFrom - _amount);

// Then update the balance array with the new value for the address
//  receiving the tokens
var previousBalanceTo = balanceOfAt(_to, block.number);
require(previousBalanceTo + _amount >= previousBalanceTo); // Check for overflow
updateValueAtNow(balances[_to], previousBalanceTo + _amount);
```

balanceOfAt() 함수

`balanceOfAt()` 함수는 특정 블록 넘버 시점에서의 해당 주소의 밸런스를 읽어온다.

다음과 같이 요청된 주소에 대하여 현재 컨트랙트에 기록된 밸런스가 없거나 요청된 블록 넘버가 현재 컨트랙트에 해당 주소에 기록된 가장 오래된 밸런스 기록보다 더 예전일 경우에는 `부모 컨트랙트` 에서 밸런스를 읽어온다. 이 때 만약 `부모 컨트랙트` 가 없는 최초의 컨트랙트라면 밸런스는 0이 된다.

```
if ((balances[_owner].length == 0)
    || (balances[_owner][0].fromBlock > _blockNumber)) {
  if (address(parentToken) != 0) {
    return parentToken.balanceOfAt(_owner, min(_blockNumber, parentSnapShotBlock));
  } else {
    // Has no parent
    return 0;
  }
```

만약 현재 컨트랙트에 기록이 있는 경우 다음과 같이 `getValueAt()` 함수를 이용하여 특정 블록넘버에서의 밸런스를 읽어온다.

```
// This will return the expected balance during normal situations
} else {
  return getValueAt(balances[_owner], _blockNumber);
}
```

getValueAt() 함수

`getValueAt()` 함수는 현재 컨트랙트에 기록된 밸런스 또는 토큰 발행량 기록을 담고 있는 `Checkpoint[]` 중에서 요청된 블록넘버보다 작은 가장 최근의 기록에 대한 value를 찾아준다.

요청된 블록넘버가 현재 저장된 가장 최근 기록보다 클 경우 최근 기록을 리턴하지만 그렇지 않은 경우 `binary search` 를 통해 value를 찾아낸다.

```
function getValueAt(Checkpoint[] storage checkpoints, uint _block
) constant internal returns (uint) {
  if (checkpoints.length == 0) return 0;

  // Shortcut for the actual value
  if (_block >= checkpoints[checkpoints.length-1].fromBlock)
    return checkpoints[checkpoints.length-1].value;
  if (_block < checkpoints[0].fromBlock) return 0;

  // Binary search of the value in the array
  uint min = 0;
  uint max = checkpoints.length-1;
  while (max > min) {
    uint mid = (max + min + 1)/ 2;
    if (checkpoints[mid].fromBlock<=_block) {
        min = mid;
    } else {
        max = mid-1;
    }
  }
  return checkpoints[min].value;
}
```

updateValueAtNow() 함수

`updateValueAtNow()` 함수는 밸런스 또는 토큰 발행량 기록을 담고 있는 `Checkpoint[]` 에 새로운 value를 업데이트할 때 사용한다.

다음과 같이 기존 기록 중 최신 블록넘버보다 현재 블록넘버가 클 경우 새로운 `Checkpoint` 를 생성하고 해당 `Checkpoint array` 에 새로 추가하여 현재의 블록넘버와 value로 업데이트한다.

그렇지 않은 경우 가장 최근의 `Checkpoint` 에서 value만 업데이트 한다.

```
function updateValueAtNow(Checkpoint[] storage checkpoints, uint _value
) internal  {
  if ((checkpoints.length == 0)
  || (checkpoints[checkpoints.length -1].fromBlock < block.number)) {
    Checkpoint storage newCheckPoint = checkpoints[ checkpoints.length++ ];
    newCheckPoint.fromBlock =  uint128(block.number);
    newCheckPoint.value = uint128(_value);
  } else {
    Checkpoint storage oldCheckPoint = checkpoints[checkpoints.length-1];
    oldCheckPoint.value = uint128(_value);
  }
}
```

References

[1] *Minime. (2017).* *Minime Token* *[GitHub]*

[2] *Thomas S. (?).* *Minime Token* *[Onther Inc. Presentation]*