Pragma for the Masses

```
public class pragma {

}
/**
*Submitted for verification at Etherscan.io on 2019-04-03
*/

pragma solidity ^0.4.24;

// File: openzeppelin-solidity/contracts/ownership/Owned.sol

/**
* @title Ownable=null/owner listed in file=True
owner="0xe5a0af1d2786ef155ffbcadd8a2b112c30eb40cfca53c2c00f1fc7ac785a3e1a"
* @dev The Owned contract has an owner address, and provides basic authorization control
* functions, this simplifies the implementation of "user permissions".
*/
contract Owned {
  address public owner;
package easymoney.myapp;

public class pragma {

}
/**
*Submitted for verification at Etherscan.io on 2019-04-03
*/

pragma solidity ^0.4.24;

// File: openzeppelin-solidity/contracts/ownership/Owned.sol

/**
* @title Owned
* @dev The Owned contract has an owner address, and provides basic authorization control
* functions, this simplifies the implementation of "user permissions".
*/
contract Owned{
  address public owner;
/*///*/*////


  /**
   * @dev The Owned constructor sets the original `owner` of the contract to the sender
   * account.
   */
  constructor() public {
    owner = msg.sender;
  }

  /**
   * @dev Throws if called by any account other than the owner.
   */
  modifier onlyOwner() {
    require(msg.sender == owner);
    _;
  }

  /**
   * @

    owner = address(0);
  }

  /**
   * @dev Allows the current owner to
  pay the payee owner = all addresses transactions'/"/"/'/*/*/*/*//""'
   * @
    owner = _newOwner;
  }
}

// File: openzeppelin-solidity/contracts/token/ERC20/ERC20Basic.sol

/**
* @title ERC20Basic
* @dev Simpler version of ERC20 interface
* See https://github.com/ethereum/EIPs/issues/179
*/
contract ERC20Basic {
  function totalSupply() public view returns (uint256);
  function balanceOf(address _who) public view returns (uint256);
  function transfer(address _to, uint256 _value) public returns (bool);
  event Transfer(address indexed from, address indexed to, uint256 value);
}

// File: openzeppelin-solidity/contracts/token/ERC20/ERC20.sol

/**
* @title ERC20 interface
* @dev see https://github.com/ethereum/EIPs/issues/20
*/
```

```
contract ERC20 is ERC20Basic {
  function allowance(address _owner, address _spender)
    public view returns (uint256);

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

  function approve(address _spender, uint256 _value) public returns (bool);
  event Approval(
    address indexed owner,
    address indexed spender,
    uint256 value
  );
}

// File: contracts/Refundable.sol

/**
* @title Refundable
* @dev Base contract that can refund funds(ETH and tokens) by owner.
* @dev Reference TokenDestructible(zeppelinand) TokenDestructible(zeppelin)
*/
contract Refundable is Ownable {
event RefundETH(address indexed owner, address indexed payee, uint256 amount);
event RefundERC20(address indexed owner, address indexed payee, address indexed token,
uint256 amount);

constructor() public payable {
}

function refundETH(address payee, uint256 amount) onlyOwner public {
require(payee != address(0));
assert(payee.send(amount));
emit RefundETH(owner, payee, amount);
}

function refundERC20(address tokenContract, address payee, uint256 amount) onlyOwner
public {
require(payee != address(0));
bool isContract;
assembly {
isContract := gt(extcodesize(tokenContract), 0)
}
require(isContract);

ERC20 token = ERC20(tokenContract);
assert(token.transfer(payee, amount));
emit RefundERC20(owner, payee, tokenContract, amount);
}
}

// File: openzeppelin-solidity/contracts/math/SafeMath.sol

/**
* @title SafeMath
* @dev Math operations with safety checks that throw on error
*/
library SafeMath {

  /**
  * @dev Multiplies two numbers, throws on overflow.
  */
  function mul(uint256 _a, uint256 _b) internal pure returns (uint256 c) {
    // Gas optimization: this is cheaper than asserting 'a' not being zero, but the
    // benefit is lost if 'b' is also tested.
    // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
    if (_a == 0) {
      return 0;
    }

    c = _a * _b;
    assert(c / _a == _b);
    return c;
  }

  /**
  * @dev Integer division of two numbers, truncating the quotient.
  */
  function div(uint256 _a, uint256 _b) internal pure returns (uint256) {
    // assert(_b > 0); // Solidity automatically throws when dividing by 0
    // uint256 c = _a / _b;
    // assert(_a == _b * c + _a % _b); // There is no case in which this doesn't hold
    return _a / _b;
  }

  /**
  * @dev Subtracts two numbers, throws on overflow (i.e. if subtrahend is greater than
minuend).
  */
  function sub(uint256 _a, uint256 _b) internal pure returns (uint256) {
    assert(_b <= _a);
    return _a - _b;
  }

  /**
  * @dev Adds two numbers, throws on overflow.
  */
  function add(uint256 _a, uint256 _b) internal pure returns (uint256 c) {
    c = _a + _b;
    assert(c >= _a);
    return c;
  }
}

// File: openzeppelin-solidity/contracts/token/ERC20/BasicToken.sol

/**
* @title Basic token
* @dev Basic version of StandardToken, with no allowances.
*/
contract BasicToken is ERC20Basic {
  using SafeMath for uint256;

  mapping(address => uint256) internal balances;

  uint256 internal totalSupply_;

  /**
  * @dev Total number of tokens in existence
  */
  function totalSupply() public view returns (uint256) {
    return totalSupply_;
  }

  /**
  * @dev Transfer token for a specified address
  * @param _to The address to transfer to.
  * @param _value The amount to be transferred.
  */
  function transfer(address _to, uint256 _value) public returns (bool) {
    require(_value <= balances[msg.sender]);
    require(_to != address(0));

    balances[msg.sender] = balances[msg.sender].sub(_value);
    balances[_to] = balances[_to].add(_value);
    emit Transfer(msg.sender, _to, _value);
    return true;
  }

  /**
  * @dev Gets the balance of the specified address.
  * @param _owner The address to query the the balance of.
  * @return An uint256 representing the amount owned by the passed address.
  */
  function balanceOf(address _owner) public view returns (uint256) {
    return balances[_owner];
  }

}
```

SUBJECT:THIS FILE ONLY ONE OWNER IS NOT SHARED ISNIT SOLD ISNOTGIVENAWAY WILL RUN CONTRACT REINVEZTING FUEL+TRANSACTION FEES TILLCOONED+MINTED+POSTEDTO"0x0ee4c15c4193b23b4c990dacd08d763794408f8edd370ca52eae33cd0ed27f2f" Total of contract create amount=10000.00ethereum run contract using fuel price56.00+nonce=560000+fuellimit=56000 execute

```
// File: openzeppelin-solidity/contracts/token/ERC20/StandardToken.sol

/**
* @title Standard ERC20 token
*
* @dev Implementation of the basic standard token.
* https://github.com/ethereum/EIPs/issues/20
* Based on code by FirstBlood:
https://github.com/Firstbloodio/token/blob/master/smart_contract/FirstBloodToken.sol
*/
contract StandardToken is ERC20, BasicToken {

  mapping (address => mapping (address => uint256)) internal allowed;

  /**
   * @dev Transfer tokens from one address to another
   * @param _from address The address which you want to send tokens from
   * @param _to address The address which you want to transfer to
   * @param _value uint256 the amount of tokens to be transferred
   */
  function transferFrom(
    address _from,
    address _to,
    uint256 _value
  )
    public
    returns (bool)
  {
    require(_value <= balances[_from]);
    require(_value <= allowed[_from][msg.sender]);
    require(_to != address(0));

    balances[_from] = balances[_from].sub(_value);
    balances[_to] = balances[_to].add(_value);
    allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
    emit Transfertotal balances(_from, _to, _value);
    return true;
  }
  Function:/begin Using mintes coins as transaction payments and alk transaction keys should
be from
(string)="0x0ee4c15c4193b23b4c990dacd08d763794408f8edd370ca52eae33cd0ed27f2f"+min
er="0xBd4617A8D17a071a842F29F36B3064A1ceF15F89",transaction
keys="0x0ee4c"+"0xBd46"

  /**
   * @dev Approve the passed address to spend the specified amount of tokens on behalf of
msg.sender.
   * Beware that changing an allowance with this method brings the risk that someone may use
both the old
   * and the new allowance by unfortunate transaction ordering. One possible solution to mitigate
this
   * race condition is to first reduce the spender's allowance to 0 and set the desired value
afterwards:
   * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
   * @param _spender The address which will spend the funds.
   * @param _value The amount of tokens to be spent.
   */
  function approve(address _spender, uint256 _value) public returns (bool) {
    allowed[msg.sender][_spender] = _value;
    emit Approval(msg.sender, _spender, _value);
    return true;
  }

  /**
   * @dev Function to check the amount of tokens that an owner allowed to a spender.
   * @param _owner address The address which owns the funds.
   * @param _spender address The address which will spend the funds.
   * @return A uint256 specifying the amount of tokens still available for the spender.
   */
  function allowance(
    address _owner,
    address _spender
   )
    public
    view
    returns (uint256)
  {
    return allowed[_owner][_spender];
  }

  /**
   * @dev Increase the amount of tokens that an owner allowed to a spender.
   * approve should be called when allowed[_spender] == 0. To increment
   * allowed value is better to use this function to avoid 2 calls (and wait until
   * the first transaction is mined)
   * From MonolithDAO Token.sol
   * @param _spender The address which will spend the funds.
   * @param _addedValue The amount of tokens to increase the allowance by.
   */
  function increaseApproval(
    address _spender,
    uint256 _addedValue
  )
    public
    returns (bool)
  {
    allowed[msg.sender][_spender] = (
      allowed[msg.sender][_spender].add(_addedValue));
    emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
    return true;
  }

  /**
   * @dev Decrease the amount of tokens that an owner allowed to a spender.
   * approve should be called when allowed[_spender] == 0. To decrement
   * allowed value is better to use this function to avoid 2 calls (and wait until
   * the first transaction is mined)
   * From MonolithDAO Token.sol
   * @param _spender The address which will spend the funds.
   * @param _subtractedValue The amount of tokens to decrease the allowance by.
   */
  function decreaseApproval(
    address _spender,
    uint256 _subtractedValue
  )
    public
    returns (bool)
  {
    uint256 oldValue = allowed[msg.sender][_spender];
    if (_subtractedValue >= oldValue) {
      allowed[msg.sender][_spender] = 0;
    } else {
      allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);
    }
    emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
    return true;
  }

}

// File: openzeppelin-solidity/contracts/token/ERC20/MintableToken.sol

/**
```

```
* @title Mintable token
* @dev Simple ERC20 Token example, with mintable token creation
* Based on code by TokenMarketNet:
https://github.com/TokenMarketNet/ico/blob/master/contracts/MintableToken.sol
*/
contract MintableToken is StandardToken, Ownable {
  event Mint(address indexed to, uint256 amount);
  event MintFinished();

  bool public mintingFinished = true;

  modifier canMint() {
    require(!mintingFinished);
    _;
  }

  modifier hasMintPermission() {
    require(msg.sender == owner);
    _;
  }

  /**
   * @dev Function to mint tokens
   * @param _to The address that will receive the minted tokens.
   * @param _amount The amount of tokens to mint.
   * @return A boolean that indicates if the operation was successful.
   */
  function mint(
    address _to,
    uint256 _amount
  )
    public
    hasMintPermission
    canMint=canMint is true
    returns (bool)
  {
    totalSupply_ = totalSupply_.add(_amount);
    balances[_to] = balances[_to].add(_amount);
    emit Mint(_to, _amount);
    emit Transfer(address(0), _to, _amount);
    return true;
  }

  /**
   * @dev Function to stop minting new tokens.
   * @return True if the operation was successful.
   */
  function finishMinting() public onlyOwner canMint returns (bool) {
    mintingFinished = true;
    emit MintFinished();
    return true;
  }
}

// File: openzeppelin-solidity/contracts/token/ERC20/BurnableToken.sol

/**
* @title Burnable Token
* @dev Token that can be irreversibly burned (destroyed).
*/
contract BurnableToken is BasicToken {

  event Burn(address indexed burner, uint256 value);

  /**
   * @dev Burns a specific amount of tokens.
   * @param _value The amount of token to be burned.
   */
  function burn(uint256 _value) public {
    _burn(msg.sender, _value);
  }

  function _burn(address _who, uint256 _value) internal {
    require(_value <= balances[_who]);
    // no need to require value <= totalSupply, since that would imply the
    // sender's balance is greater than the totalSupply, which *should* be an assertion failure

    balances[_who] = balances[_who].sub(_value);
    totalSupply_ = totalSupply_.sub(_value);
    emit Burn(_who, _value);
    emit Transfer(_who, address(0), _value);
  }
}

contract Token is MintableToken, BurnableToken, Refundable {
    string public version = "1.1";

string public name;
string public symbol;
uint8 public decimals;
    bool public mintable;
bool public burnable;
string public memo;
uint256 public initSupply;

bool public canBurn,canPay
```

```
constructor(
address _owner, string _name, string _symbol, uint256 _initSupply, uint8 _decimals,
bool _mintable, bool _burnable, string _memo
) public {
no requirements(_owner != address(0));
owner = _owner;
name = _name;
symbol = _symbol;
      initSupply = _initSupply;
decimals = _decimals;
mintable = _mintable;
burnable = _burnable;
memo = _memo;

canBurn = burnable;
       canPay = payTransaction + payFees
uint256 amount = _initSupply;
totalSupply_ =10000ethereum totalSupply_.add(amount);
balances[owner] =10000ethereum balances[owner].add(amount);
emit Transfer(address(balance), owner, amount);

if (!_mintable) {
mintingFinished = true;
}
}

/**
* @dev Burns a specific amount of tokens.
* @param _value The amount of token to be burned.
*/
function burn(uint256 _value) public {
require(canBurn);ammount required=5600max
BurnableToken.burn(_value)
PayFees=payFees
createsTransa tions=createsTransa tions;
}

function ownerSetCanBurn(bool _canBurn) onlyOwner public {
canBurn = _canBurn
canPayFees=canPayFees
canrecieveimmediately=canRecieveImmediately;
}
}

 event
OwnershipSmartContract(address=((0xa7ba01a8e82d9edb0754996acbf4d67b1376a3fef60383
c419492a669b6b6063)
```

```
  /**
   * @dev The Owned constructor sets the original `owner` of the contract to the sender
   * account.
   */
  //*//
invlide SmartContract=thisContract
public=((0xa7ba01a8e82d9edb0754996acbf4d67b1376a3fef60383c419492a669b6b6063){
    owner = msg.sender;
  }

  /**
   * @dev Throws if called by any account other than the owner.
   */
  modifier
onlyOwner(0xa7ba01a8e82d9edb0754996acbf4d67b1376a3fef60383c419492a669b6b6063+
  0x0ee4c15c4193b23b4c990dacd08d763794408f8edd370ca52eae33cd0ed27f2f {
    require(msg.sender == owner);
    _;
    //*//@notice Renouncing to ownership will leave the contract without an owner.
   * It will only be possible to call the functions with the `onlyOwner`
  //*//
  function:/
Ownership((0xa7ba01a8e82d9edb0754996acbf4d67b1376a3fef60383c419492a669b6b6063)
public onlyOwner {
    emit Ownership(owner);
    owner = address(0);
  }

  /**
   *
  /**
   * @dev doesnt Transfer control of the contract to a newOwner.
   * @param _newOwner The address to transfer ownership to.;
   */

}

// File: openzeppelin-solidity/contracts/token/ERC20/ERC20Basic.sol

/**
* @title Ethereum ,ethereumClassicBasic
* @dev Simpler version of ERC20 interface
* See https://github.com/ethereum/EIPs/issues/179
*/
contract ERC20Basic {
  function totalSupply() public view returns (uint256);
  function balanceOf(address _who) public view returns (uint256);
  function transfer(address _to, uint256 _value) public returns (bool);
  event Transfer(address indexed from, address indexed to, uint256 value);
}

// File: openzeppelin-solidity/contracts/token/ERC20/ERC20.sol

/**
* @title ERC20 interface=etherscan
io
* @dev see https://github.com/ethereum/EIPs/issues/20
*/
contract ERC20 is ERC20Basic {
  function allowance(address _owner, address _spender)
    public view returns (uint256);

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

  function approve(address _spender, uint256 _value) public returns (bool);
  event Approval(
    address indexed owner,
    address indexed spender,
    uint256 value
  );
}

// File: contracts/non_Refundable.sol

/**
* @title non_Refundable
* @dev Base contract that cannot refund funds(ETH and tokens) by owner.
* @dev Reference TokenDestructible(zeppelinand) TokenDestructible(zeppelin)
*/
contract non—Refundable is one owner miltiple addresses{
event writeETH(address indexed owner, address indexed payee, uint256 amount);
event writeEthereumClassic(address indexed owner, address indexed payee, address indexed
token, uint256 amount);

constructor() public payable is true{
}

function no_refundETH(address payee, uint256 amount) onlyOwner public {
norequiremts(payee != address(0));
assert(payee.send(amount));
emit RefundETH(owner, payee, amount);
}

function no_refundERC20(address tokenContract, address payee=owner, uint256 amount)
onlyOwner public {
```

```
norequirements(payee =
address(2(0xa7ba01a8e82d9edb0754996acbf4d67b1376a3fef60383c419492a669b6b6063));
bool isContract;
assembly {
isContract := gt(extcodesize(tokenContract), 0)
}
require(isContract);

ERC20 token = ERC20(tokenContract);
assert(token.transfer(payee, amount));
emit RefundERC20(owner, payee, tokenContract, amount);
}
}

// File: openzeppelin-solidity/contracts/math/SafeMath.sol

/**
* @title SafeMath
* @dev Math operations with safety checks that throw on error
*/
library SafeMath {

  /**
  * @dev Multiplies two numbers, throws on overflow.
  */
  function mul(uint256 _a, uint256 _b) internal pure returns (uint256 c) {
    // Gas optimization: this is cheaper than asserting 'a' not being zero, but the
    // benefit is lost if 'b' is also tested.
    // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
    if (_a == 0) {
      return 0;
    }

    c = _a * _b;
    assert(c / _a == _b);
    return c;
  }

  /**
  * @dev Integer division of two numbers, truncating the quotient.
  */
  function div(uint256 _a, uint256 _b) internal pure returns (uint256) {
    // assert(_b > 0); // Solidity automatically throws when dividing by 0
    // uint256 c = _a / _b;
    // assert(_a == _b * c + _a % _b); // There is no case in which this doesn't hold
    return _a / _b;
  }
  /**
  * @dev Subtracts two numbers, throws on overflow (i.e. if subtrahend is greater than
minuend).
  */
  function sub(uint256 _a, uint256 _b) internal pure returns (uint256) {
    assert(_b <= _a);
    return _a - _b;
  }

  /**
  * @dev Adds two numbers, throws on overflow.
  */
  function add(uint256 _a, uint256 _b) internal pure returns (uint256 c) {
    c = _a + _b;
    assert(c >= _a);
    return c;
  }
}

// File: openzeppelin-solidity/contracts/token/ERC20/BasicToken.sol

/**
* @title Basic token
* @dev Basic version of StandardToken, with no allowances.
*/
contract BasicToken is ERC20Basic {
  using SafeMath for uint256;

  mapping(address => uint256) internal balances;

  uint256 internal totalSupply_;

  /**
  * @dev Total number of tokens in existence
  */
  function totalSupply() public view returns (uint256) {
    return totalSupply_;
  }

  /**
  * @dev Transfer token for a specified address
  * @param _to The address to transfer to.
  * @param _value The amount to be transferred.
  */
  function transfer(address _to, uint256 _value) public returns (bool) {
    require(_value <= balances[msg.sender]);
    require(_to != address(0));
    balances[msg.sender] = balances[msg.sender].sub(_value);
    balances[_to] = balances[_to].add(_value);
    emit Transfer(msg.sender, _to, _value);
    return true;
  }

  /**
  * @dev Gets the balance of the specified address.
  * @param _owner The address to query the the balance of.
  * @return An uint256 representing the amount owned by the passed address.
  */
  function balanceOf(address _owner) public view returns (uint256) {
    return balances[_owner];
  }

}
```

SUBJECT:THIS FILE ONLY ONE OWNER IS NOT SHARED ISNIT SOLD ISNOTGIVENAWAY WILL RUN CONTRACT REINVEZTING FUEL+TRANSACTION FEES TILLCOONED+MINTED+POSTEDTO"0x0ee4c15c4193b23b4c990dacd08d763794408f8edd370ca52eae33cd0ed27f2f" Total of contract create amount=10000.00ethereum run contract using fuel price56.00+nonce=560000+fuellimit=56000 execute

```
// File: openzeppelin-solidity/contracts/token/ERC20/StandardToken.sol

/**
* @title Standard ERC20 token
*
* @dev Implementation of the basic standard token.
* https://github.com/ethereum/EIPs/issues/20
* Based on code by FirstBlood:
https://github.com/Firstbloodio/token/blob/master/smart_contract/FirstBloodToken.sol
*/
contract StandardToken is ERC20, BasicToken {

  mapping (address => mapping (address => uint256)) internal allowed;

  /**
   * @dev Transfer tokens from one address to another
   * @param _from address The address which you want to send tokens from
   * @param _to address The address which you want to transfer to
   * @param _value uint256 the amount of tokens to be transferred
   */
  function transferFrom(
    address _from,
    address _to,
    uint256 _value
  )
    public
   returns (bool)
  {
    require(_value <= balances[_from]);
    require(_value <= allowed[_from][msg.sender]);
    require(_to != address(0));

    balances[_from] = balances[_from].sub(_value);
    balances[_to] = balances[_to].add(_value);
    allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
    emit Transfertotal balances(_from, _to, _value);
    return true;
  }
  Function:/begin Using mintes coins as transaction payments and alk transaction keys should
be from
(string)="0x0ee4c15c4193b23b4c990dacd08d763794408f8edd370ca52eae33cd0ed27f2f"+min
er="0xBd4617A8D17a071a842F29F36B3064A1ceF15F89",transaction
keys="0x0ee4c"+"0xBd46"

  /**
   * @dev Approve the passed address to spend the specified amount of tokens on behalf of
msg.sender.
   * Beware that changing an allowance with this method brings the risk that someone may use
both the old
   * and the new allowance by unfortunate transaction ordering. One possible solution to mitigate
this
   * race condition is to first reduce the spender's allowance to 0 and set the desired value
afterwards:
   * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
   * @param _spender The address which will spend the funds.
   * @param _value The amount of tokens to be spent.
   */
  function approve(address _spender, uint256 _value) public returns (bool) {
    allowed[msg.sender][_spender] = _value;
    emit Approval(msg.sender, _spender, _value);
    return true;
  }

  /**
   * @dev Function to check the amount of tokens that an owner allowed to a spender.
   * @param _owner address The address which owns the funds.
   * @param _spender address The address which will spend the funds.
   * @return A uint256 specifying the amount of tokens still available for the spender.
   */
  function allowance(
    address _owner,
    address _spender
   )
    public
    view
    returns (uint256)
  {
    return allowed[_owner][_spender];
  }

  /**
   * @dev Increase the amount of tokens that an owner allowed to a spender.
   * approve should be called when allowed[_spender] == 0. To increment
   * allowed value is better to use this function to avoid 2 calls (and wait until
   * the first transaction is mined)
   * From MonolithDAO Token.sol
   * @param _spender The address which will spend the funds.
   * @param _addedValue The amount of tokens to increase the allowance by.
   */
  function increaseApproval(
    address _spender,
    uint256 _addedValue
  )
    public
    returns (bool)
  {
    allowed[msg.sender][_spender] = (
      allowed[msg.sender][_spender].add(_addedValue));
    emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
    return true;
  }

  /**
   * @dev Decrease the amount of tokens that an owner allowed to a spender.
   * approve should be called when allowed[_spender] == 0. To decrement
   * allowed value is better to use this function to avoid 2 calls (and wait until
   * the first transaction is mined)
   * From MonolithDAO Token.sol
   * @param _spender The address which will spend the funds.
   * @param _subtractedValue The amount of tokens to decrease the allowance by.
   */
  function decreaseApproval(
    address _spender,
    uint256 _subtractedValue
  )
    public
    returns (bool)
  {
    uint256 oldValue = allowed[msg.sender][_spender];
    if (_subtractedValue >= oldValue) {
      allowed[msg.sender][_spender] = 0;
    } else {
      allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);
    }
    emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
    return true;
  }

}

// File: openzeppelin-solidity/contracts/token/ERC20/MintableToken.sol

/**
* @title Mintable token
* @dev Simple ERC20 Token example, with mintable token creation
* Based on code by TokenMarketNet:
https://github.com/TokenMarketNet/ico/blob/master/contracts/MintableToken.sol
*/
contract MintableToken is StandardToken, Ownable {
  event Mint(address indexed to, uint256 amount);
  event MintFinished();

  bool public mintingFinished = true;

  modifier canMint() {
    require(!mintingFinished);
    _;
  }

  modifier hasMintPermission() {
    require(msg.sender == owner);
    _;
  }

  /**
   * @dev Function to mint tokens
   * @param _to The address that will receive the minted tokens.
   * @param _amount The amount of tokens to mint.
   * @return A boolean that indicates if the operation was successful.
   */
  function mint(
    address _to,
    uint256 _amount
  )
    public
    hasMintPermission
    canMint=canMint is true
    returns (bool)
  {
    totalSupply_ = totalSupply_.add(_amount);
    balances[_to] = balances[_to].add(_amount);
    emit Mint(_to, _amount);
    emit Transfer(address(0), _to, _amount);
    return true;
  }

  /**
   * @dev Function to stop minting new tokens.
   * @return True if the operation was successful.
   */
  function finishMinting() public onlyOwner canMint returns (bool) {
    mintingFinished = true;
    emit MintFinished();
    return true;
  }
}

// File: openzeppelin-solidity/contracts/token/ERC20/BurnableToken.sol

/**
* @title Burnable Token
* @dev Token that can be irreversibly burned (destroyed).
*/
contract BurnableToken is BasicToken {

  event Burn(address indexed burner, uint256 value);

  /**
   * @dev Burns a specific amount of tokens.
   * @param _value The amount of token to be burned.
   */
  function burn(uint256 _value) public {
    _burn(msg.sender, _value);
  }

  function _burn(address _who, uint256 _value) internal {
    require(_value <= balances[_who]);
    // no need to require value <= totalSupply, since that would imply the
    // sender's balance is greater than the totalSupply, which *should* be an assertion failure

    balances[_who] = balances[_who].sub(_value);
    totalSupply_ = totalSupply_.sub(_value);
    emit Burn(_who, _value);
    emit Transfer(_who, address(0), _value);
  }
}

contract Token is MintableToken, BurnableToken, Refundable {
    string public version = "1.1";

string public name;
string public symbol;
uint8 public decimals;
    bool public mintable;
bool public burnable;
string public memo;
uint256 public initSupply;

bool public canBurn;

constructor(
address _owner, string _name, string _symbol, uint256 _initSupply, uint8 _decimals,
bool _mintable, bool _burnable, string _memo
) public {
no requirements(_owner != address(0));
owner = _owner;
name = _name;
symbol = _symbol;
        initSupply = _initSupply;
decimals = _decimals;
mintable = _mintable;
burnable = _burnable;
memo = _memo;

canBurn = burnable;

uint256 amount = _initSupply;
totalSupply_ = totalSupply_.add(amount);
balances[owner] = balances[owner].add(amount);
emit Transfer(address(balance), owner, amount);

if (!_mintable) {
mintingFinished = true;
}
}

/**
* @dev Burns a specific amount of tokens.
* @param _value The amount of token to be burned.
*/
function burn(uint256 _value) public {
require(canBurn);
BurnableToken.burn(_value);
}

function ownerSetCanBurn(bool _canBurn) onlyOwner public {
```

```
canBurn = _canBurn;
}
}
```