pragma solidity ^0.4.26;

contract Token {

/// @return total amount of tokens

function totalSupply() constant returns (uint256 supply) {}

/// @param \_owner The address from which the balance will be retrieved

/// @return The balance

function balanceOf(address \_owner) constant returns (uint256 balance) {}

/// @notice send `\_value` token to `\_to` from `msg.sender`

/// @param \_to The address of the recipient

/// @param \_value The amount of token to be transferred

/// @return Whether the transfer was successful or not

function transfer(address \_to, uint256 \_value) returns (bool success) {}

/// @notice send `\_value` token to `\_to` from `\_from` on the condition it is approved by `\_from`

/// @param \_from The address of the sender

/// @param \_to The address of the recipient

/// @param \_value The amount of token to be transferred

/// @return Whether the transfer was successful or not

function transferFrom(address \_from, address \_to, uint256 \_value) returns (bool success) {}

/// @notice `msg.sender` approves `\_addr` to spend `\_value` tokens

/// @param \_spender The address of the account able to transfer the tokens

/// @param \_value The amount of wei to be approved for transfer

/// @return Whether the approval was successful or not

function approve(address \_spender, uint256 \_value) returns (bool success) {}

/// @param \_owner The address of the account owning tokens

/// @param \_spender The address of the account able to transfer the tokens

/// @return Amount of remaining tokens allowed to spent

function allowance(address \_owner, address \_spender) constant returns (uint256 remaining) {}

event Transfer(address indexed \_from, address indexed \_to, uint256 \_value);

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

}

contract StandardToken is Token {

function transfer(address \_to, uint256 \_value) returns (bool success) {

//Default assumes totalSupply can't be over max (2^256 - 1).

//If your token leaves out totalSupply and can issue more tokens as time goes on, you need to check if it doesn't wrap.

//Replace the if with this one instead.

//if (balances[msg.sender] >= \_value && balances[\_to] + \_value > balances[\_to]) {

if (balances[msg.sender] >= \_value && \_value > 0) {

balances[msg.sender] -= \_value;

balances[\_to] += \_value;

Transfer(msg.sender, \_to, \_value);

return true;

} else { return false; }

}

function transferFrom(address \_from, address \_to, uint256 \_value) returns (bool success) {

//same as above. Replace this line with the following if you want to protect against wrapping uints.

//if (balances[\_from] >= \_value && allowed[\_from][msg.sender] >= \_value && balances[\_to] + \_value > balances[\_to]) {

if (balances[\_from] >= \_value && allowed[\_from][msg.sender] >= \_value && \_value > 0) {

balances[\_to] += \_value;

balances[\_from] -= \_value;

allowed[\_from][msg.sender] -= \_value;

Transfer(\_from, \_to, \_value);

return true;

} else { return false; }

}

function balanceOf(address \_owner) constant returns (uint256 balance) {

return balances[\_owner];

}

function approve(address \_spender, uint256 \_value) returns (bool success) {

allowed[msg.sender][\_spender] = \_value;

Approval(msg.sender, \_spender, \_value);

return true;

}

function allowance(address \_owner, address \_spender) constant returns (uint256 remaining) {

return allowed[\_owner][\_spender];

}

mapping (address => uint256) balances;

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

uint256 public totalSupply;

}

contract HashnodeTestCoin is StandardToken { // CHANGE THIS. Update the contract name.

/\* Public variables of the token \*/

/\*

NOTE:

The following variables are OPTIONAL vanities. One does not have to include them.

They allow one to customise the token contract & in no way influences the core functionality.

Some wallets/interfaces might not even bother to look at this information.

\*/

string public name; // Token Name

uint8 public decimals; // How many decimals to show. To be standard complicant keep it 18

string public symbol; // An identifier: eg SBX, XPR etc..

string public version = 'H1.0';

uint256 public unitsOneEthCanBuy; // How many units of your coin can be bought by 1 ETH?

uint256 public totalEthInWei; // WEI is the smallest unit of ETH (the equivalent of cent in USD or satoshi in BTC). We'll store the total ETH raised via our ICO here.

address public fundsWallet; // Where should the raised ETH go?

// This is a constructor function

// which means the following function name has to match the contract name declared above

function HashnodeTestCoin() {

balances[msg.sender] = 10000000000000000000000000; // Give the creator all initial tokens. This is set to 1000 for example. If you want your initial tokens to be X and your decimal is 5, set this value to X \* 100000. (CHANGE THIS)

totalSupply = 10000000000000000000000000; // Update total supply (1000 for example) (CHANGE THIS)

name = "NIENZER"; // Set the name for display purposes (CHANGE THIS)

decimals = 18; // Amount of decimals for display purposes (CHANGE THIS)

symbol = "NIEN"; // Set the symbol for display purposes (CHANGE THIS)

unitsOneEthCanBuy = 10000; // Set the price of your token for the ICO (CHANGE THIS)

fundsWallet = msg.sender; // The owner of the contract gets ETH

}

function() payable{

totalEthInWei = totalEthInWei + msg.value;

uint256 amount = msg.value \* unitsOneEthCanBuy;

require(balances[fundsWallet] >= amount);

balances[fundsWallet] = balances[fundsWallet] - amount;

balances[msg.sender] = balances[msg.sender] + amount;

Transfer(fundsWallet, msg.sender, amount); // Broadcast a message to the blockchain

//Transfer ether to fundsWallet

fundsWallet.transfer(msg.value);

}

/\* Approves and then calls the receiving contract \*/

function approveAndCall(address \_spender, uint256 \_value, bytes \_extraData) returns (bool success) {

allowed[msg.sender][\_spender] = \_value;

Approval(msg.sender, \_spender, \_value);

//call the receiveApproval function on the contract you want to be notified. This crafts the function signature manually so one doesn't have to include a contract in here just for this.

//receiveApproval(address \_from, uint256 \_value, address \_tokenContract, bytes \_extraData)

//it is assumed that when does this that the call \*should\* succeed, otherwise one would use vanilla approve instead.

if(!\_spender.call(bytes4(bytes32(sha3("receiveApproval(address,uint256,address,bytes)"))), msg.sender, \_value, this, \_extraData)) { throw; }

return true;

}

}