// SPDX-License-Identifier: MIT

pragma solidity 0.6.12;

import "./Context.sol";

import "./IERC20.sol";

import "./SafeMath.sol";

import "./Address.sol";

import "./Ownable.sol";

contract TaxableTeamToken is IERC20, Context, Ownable {

using SafeMath for uint256;

using Address for address;

mapping (address => uint256) private \_rOwned;

mapping (address => uint256) private \_tOwned;

mapping (address => mapping (address => uint256)) private \_allowances;

mapping (address => bool) private \_isExcluded;

address[] private \_excluded;

uint256 private constant MAX = ~uint256(0);

uint256 private \_tTotal;

uint256 private \_rTotal;

uint256 private \_tFeeTotal;

string private \_name;

string private \_symbol;

uint8 private \_decimals;

uint256 private \_feesPercentage;

modifier checkIsAddressValid(address ethAddress)

{

require(ethAddress != address(0), "[Validation] invalid address");

require(ethAddress == address(ethAddress), "[Validation] invalid address");

\_;

}

modifier checkIsFeesValid(uint256 fees)

{

require(fees > 0, "[Validation] fees should be greater than zero.");

require(fees < 21, "[Validation] fees should be less than 21.");

\_;

}

constructor(

string memory name,

string memory symbol,

uint8 decimals,

uint256 supply,

uint256 fees,

address owner,

address feeWallet

) public checkIsFeesValid(fees) checkIsAddressValid(owner) checkIsAddressValid(feeWallet) {

require(decimals >=8 && decimals <= 18, "[Validation] Not valid decimals");

require(supply > 0, "[Validation] inital supply should be greater than 0");

require(owner != feeWallet, "[Validation] fee wallet and owner wallet cannot be same.");

\_name = name;

\_symbol = symbol;

\_decimals = decimals;

\_feesPercentage = fees;

\_tTotal = supply;

\_rTotal = (MAX - (MAX % \_tTotal));

\_rOwned[owner] = \_rTotal.div(1000).mul(995);

\_rOwned[feeWallet] = \_rTotal.div(1000).mul(5);

emit Transfer(address(0), owner, \_tTotal\*995/1000);

emit Transfer(address(0), feeWallet, \_tTotal\*5/1000);

}

function name() public view returns (string memory) {

return \_name;

}

function symbol() public view returns (string memory) {

return \_symbol;

}

function decimals() public view returns (uint8) {

return \_decimals;

}

function totalSupply() public view override returns (uint256) {

return \_tTotal;

}

function fees() public view returns (uint256) {

return \_feesPercentage;

}

function balanceOf(address account) public view override returns (uint256) {

if (\_isExcluded[account]) return \_tOwned[account];

return tokenFromReflection(\_rOwned[account]);

}

function transfer(address recipient, uint256 amount) public override returns (bool) {

\_transfer(\_msgSender(), recipient, amount);

return true;

}

function allowance(address owner, address spender) public view override returns (uint256) {

return \_allowances[owner][spender];

}

function approve(address spender, uint256 amount) public override returns (bool) {

\_approve(\_msgSender(), spender, amount);

return true;

}

function transferFrom(address sender, address recipient, uint256 amount) public override returns (bool) {

\_transfer(sender, recipient, amount);

\_approve(sender, \_msgSender(), \_allowances[sender][\_msgSender()].sub(amount, "ERC20: transfer amount exceeds allowance"));

return true;

}

function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {

\_approve(\_msgSender(), spender, \_allowances[\_msgSender()][spender].add(addedValue));

return true;

}

function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool) {

\_approve(\_msgSender(), spender, \_allowances[\_msgSender()][spender].sub(subtractedValue, "ERC20: decreased allowance below zero"));

return true;

}

function isExcluded(address account) public view returns (bool) {

return \_isExcluded[account];

}

function totalFees() public view returns (uint256) {

return \_tFeeTotal;

}

function reflect(uint256 tAmount) public {

address sender = \_msgSender();

require(!\_isExcluded[sender], "Excluded addresses cannot call this function");

(uint256 rAmount,,,,) = \_getValues(tAmount);

\_rOwned[sender] = \_rOwned[sender].sub(rAmount);

\_rTotal = \_rTotal.sub(rAmount);

\_tFeeTotal = \_tFeeTotal.add(tAmount);

}

function reflectionFromToken(uint256 tAmount, bool deductTransferFee) public view returns(uint256) {

require(tAmount <= \_tTotal, "Amount must be less than supply");

if (!deductTransferFee) {

(uint256 rAmount,,,,) = \_getValues(tAmount);

return rAmount;

} else {

(,uint256 rTransferAmount,,,) = \_getValues(tAmount);

return rTransferAmount;

}

}

function tokenFromReflection(uint256 rAmount) public view returns(uint256) {

require(rAmount <= \_rTotal, "Amount must be less than total reflections");

uint256 currentRate = \_getRate();

return rAmount.div(currentRate);

}

function excludeAccount(address account) external onlyOwner() {

require(!\_isExcluded[account], "Account is already excluded");

if(\_rOwned[account] > 0) {

\_tOwned[account] = tokenFromReflection(\_rOwned[account]);

}

\_isExcluded[account] = true;

\_excluded.push(account);

}

function setFeesPercentage(uint256 feesPercentage) external onlyOwner() checkIsFeesValid(feesPercentage) {

\_feesPercentage = feesPercentage;

}

function includeAccount(address account) external onlyOwner() {

require(\_isExcluded[account], "Account is already excluded");

for (uint256 i = 0; i < \_excluded.length; i++) {

if (\_excluded[i] == account) {

\_excluded[i] = \_excluded[\_excluded.length - 1];

\_tOwned[account] = 0;

\_isExcluded[account] = false;

\_excluded.pop();

break;

}

}

}

function \_approve(address owner, address spender, uint256 amount) private {

require(owner != address(0), "ERC20: approve from the zero address");

require(spender != address(0), "ERC20: approve to the zero address");

\_allowances[owner][spender] = amount;

emit Approval(owner, spender, amount);

}

function \_transfer(address sender, address recipient, uint256 amount) private {

require(sender != address(0), "ERC20: transfer from the zero address");

require(recipient != address(0), "ERC20: transfer to the zero address");

require(amount > 0, "Transfer amount must be greater than zero");

if (\_isExcluded[sender] && !\_isExcluded[recipient]) {

\_transferFromExcluded(sender, recipient, amount);

} else if (!\_isExcluded[sender] && \_isExcluded[recipient]) {

\_transferToExcluded(sender, recipient, amount);

} else if (!\_isExcluded[sender] && !\_isExcluded[recipient]) {

\_transferStandard(sender, recipient, amount);

} else if (\_isExcluded[sender] && \_isExcluded[recipient]) {

\_transferBothExcluded(sender, recipient, amount);

} else {

\_transferStandard(sender, recipient, amount);

}

}

function \_transferStandard(address sender, address recipient, uint256 tAmount) private {

(uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee) = \_getValues(tAmount);

\_rOwned[sender] = \_rOwned[sender].sub(rAmount);

\_rOwned[recipient] = \_rOwned[recipient].add(rTransferAmount);

\_reflectFee(rFee, tFee);

emit Transfer(sender, recipient, tTransferAmount);

}

function \_transferToExcluded(address sender, address recipient, uint256 tAmount) private {

(uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee) = \_getValues(tAmount);

\_rOwned[sender] = \_rOwned[sender].sub(rAmount);

\_tOwned[recipient] = \_tOwned[recipient].add(tTransferAmount);

\_rOwned[recipient] = \_rOwned[recipient].add(rTransferAmount);

\_reflectFee(rFee, tFee);

emit Transfer(sender, recipient, tTransferAmount);

}

function \_transferFromExcluded(address sender, address recipient, uint256 tAmount) private {

(uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee) = \_getValues(tAmount);

\_tOwned[sender] = \_tOwned[sender].sub(tAmount);

\_rOwned[sender] = \_rOwned[sender].sub(rAmount);

\_rOwned[recipient] = \_rOwned[recipient].add(rTransferAmount);

\_reflectFee(rFee, tFee);

emit Transfer(sender, recipient, tTransferAmount);

}

function \_transferBothExcluded(address sender, address recipient, uint256 tAmount) private {

(uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee) = \_getValues(tAmount);

\_tOwned[sender] = \_tOwned[sender].sub(tAmount);

\_rOwned[sender] = \_rOwned[sender].sub(rAmount);

\_tOwned[recipient] = \_tOwned[recipient].add(tTransferAmount);

\_rOwned[recipient] = \_rOwned[recipient].add(rTransferAmount);

\_reflectFee(rFee, tFee);

emit Transfer(sender, recipient, tTransferAmount);

}

function \_reflectFee(uint256 rFee, uint256 tFee) private {

\_rTotal = \_rTotal.sub(rFee);

\_tFeeTotal = \_tFeeTotal.add(tFee);

}

function \_getValues(uint256 tAmount) private view returns (uint256, uint256, uint256, uint256, uint256) {

(uint256 tTransferAmount, uint256 tFee) = \_getTValues(tAmount);

uint256 currentRate = \_getRate();

(uint256 rAmount, uint256 rTransferAmount, uint256 rFee) = \_getRValues(tAmount, tFee, currentRate);

return (rAmount, rTransferAmount, rFee, tTransferAmount, tFee);

}

function \_getTValues(uint256 tAmount) private view returns (uint256, uint256) {

uint256 tFee = tAmount.mul(\_feesPercentage).div(100);

uint256 tTransferAmount = tAmount.sub(tFee);

return (tTransferAmount, tFee);

}

function \_getRValues(uint256 tAmount, uint256 tFee, uint256 currentRate) private pure returns (uint256, uint256, uint256) {

uint256 rAmount = tAmount.mul(currentRate);

uint256 rFee = tFee.mul(currentRate);

uint256 rTransferAmount = rAmount.sub(rFee);

return (rAmount, rTransferAmount, rFee);

}

function \_getRate() private view returns(uint256) {

(uint256 rSupply, uint256 tSupply) = \_getCurrentSupply();

return rSupply.div(tSupply);

}

function \_getCurrentSupply() private view returns(uint256, uint256) {

uint256 rSupply = \_rTotal;

uint256 tSupply = \_tTotal;

for (uint256 i = 0; i < \_excluded.length; i++) {

if (\_rOwned[\_excluded[i]] > rSupply || \_tOwned[\_excluded[i]] > tSupply) return (\_rTotal, \_tTotal);

rSupply = rSupply.sub(\_rOwned[\_excluded[i]]);

tSupply = tSupply.sub(\_tOwned[\_excluded[i]]);

}

if (rSupply < \_rTotal.div(\_tTotal)) return (\_rTotal, \_tTotal);

return (rSupply, tSupply);

}

}