pragma solidity ^0.4.20; // solhint-disable-line

/// @title Interface for contracts conforming to ERC-721: Non-Fungible Tokens

/// @author Dieter Shirley <dete@axiomzen.co> (https://github.com/dete)

contract ERC721 {

// Required methods

function approve(address \_to, uint256 \_tokenId) public;

function balanceOf(address \_owner) public view returns (uint256 balance);

function implementsERC721() public pure returns (bool);

function ownerOf(uint256 \_tokenId) public view returns (address addr);

function takeOwnership(uint256 \_tokenId) public;

function totalSupply() public view returns (uint256 total);

function transferFrom(address \_from, address \_to, uint256 \_tokenId) public;

function transfer(address \_to, uint256 \_tokenId) public;

event Transfer(address indexed from, address indexed to, uint256 tokenId);

event Approval(address indexed owner, address indexed approved, uint256 tokenId);

// Optional

// function name() public view returns (string name);

// function symbol() public view returns (string symbol);

// function tokenOfOwnerByIndex(address \_owner, uint256 \_index) external view returns (uint256 tokenId);

// function tokenMetadata(uint256 \_tokenId) public view returns (string infoUrl);

}

contract PokemonToken is ERC721 {

/\*\*\* EVENTS \*\*\*/

/// @dev The Birth event is fired whenever a new pokemon comes into existence.

event Birth(uint256 tokenId, string name, address owner);

/// @dev The TokenSold event is fired whenever a token is sold.

event TokenSold(uint256 tokenId, uint256 oldPrice, uint256 newPrice, address prevOwner, address winner, string name);

/// @dev Transfer event as defined in current draft of ERC721.

/// ownership is assigned, including births.

event Transfer(address from, address to, uint256 tokenId);

/\*\*\* CONSTANTS \*\*\*/

/// @notice Name and symbol of the non fungible token, as defined in ERC721.

string public constant NAME = "CryptoPokemons"; // solhint-disable-line

string public constant SYMBOL = "PokemonToken"; // solhint-disable-line

uint256 private startingPrice = 0.001 ether;

uint256 private constant PROMO\_CREATION\_LIMIT = 6000;

uint256 private firstStepLimit = 0.053613 ether;

uint256 private secondStepLimit = 0.564957 ether;

/\*\*\* STORAGE \*\*\*/

/// @dev A mapping from pokemon IDs to the address that owns them. All pokemons have

/// some valid owner address.

mapping (uint256 => address) public pokemonIndexToOwner;

// @dev A mapping from owner address to count of tokens that address owns.

// Used internally inside balanceOf() to resolve ownership count.

mapping (address => uint256) private ownershipTokenCount;

/// @dev A mapping from PokemonIDs to an address that has been approved to call

/// transferFrom(). Each Pokemon can only have one approved address for transfer

/// at any time. A zero value means no approval is outstanding.

mapping (uint256 => address) public pokemonIndexToApproved;

// @dev A mapping from PersonIDs to the price of the token.

mapping (uint256 => uint256) private pokemonIndexToPrice;

// The addresses of the accounts (or contracts) that can execute actions within each roles.

address public ceoAddress;

address public cooAddress;

uint256 public promoCreatedCount;

/\*\*\* DATATYPES \*\*\*/

struct Pokemon {

string name;

}

Pokemon[] private pokemons;

/\*\*\* ACCESS MODIFIERS \*\*\*/

/// @dev Access modifier for CEO-only functionality

modifier onlyCEO() {

require(msg.sender == ceoAddress);

\_;

}

/// @dev Access modifier for COO-only functionality

modifier onlyCOO() {

require(msg.sender == cooAddress);

\_;

}

/// Access modifier for contract owner only functionality

modifier onlyCLevel() {

require(

msg.sender == ceoAddress ||

msg.sender == cooAddress

);

\_;

}

/\*\*\* CONSTRUCTOR \*\*\*/

function PokemonToken() public {

ceoAddress = msg.sender;

cooAddress = msg.sender;

}

/\*\*\* PUBLIC FUNCTIONS \*\*\*/

/// @notice Grant another address the right to transfer token via takeOwnership() and transferFrom().

/// @param \_to The address to be granted transfer approval. Pass address(0) to

/// clear all approvals.

/// @param \_tokenId The ID of the Token that can be transferred if this call succeeds.

/// @dev Required for ERC-721 compliance.

function approve(

address \_to,

uint256 \_tokenId

) public {

// Caller must own token.

require(\_owns(msg.sender, \_tokenId));

pokemonIndexToApproved[\_tokenId] = \_to;

Approval(msg.sender, \_to, \_tokenId);

}

/// For querying balance of a particular account

/// @param \_owner The address for balance query

/// @dev Required for ERC-721 compliance.

function balanceOf(address \_owner) public view returns (uint256 balance) {

return ownershipTokenCount[\_owner];

}

/// @dev Creates a new promo Pokemon with the given name, with given \_price and assignes it to an address.

function createPromoPokemon(address \_owner, string \_name, uint256 \_price) public onlyCOO {

require(promoCreatedCount < PROMO\_CREATION\_LIMIT);

address pokemonOwner = \_owner;

if (pokemonOwner == address(0)) {

pokemonOwner = cooAddress;

}

if (\_price <= 0) {

\_price = startingPrice;

}

promoCreatedCount++;

\_createPokemon(\_name, pokemonOwner, \_price);

}

/// @dev Creates a new Pokemon with the given name.

function createContractPokemon(string \_name) public onlyCOO {

\_createPokemon(\_name, address(this), startingPrice);

}

/// @notice Returns all the relevant information about a specific pokemon.

/// @param \_tokenId The tokenId of the person of interest.

function getPokemon(uint256 \_tokenId) public view returns (

string pokemonName,

uint256 sellingPrice,

address owner

) {

Pokemon storage pokemon = pokemons[\_tokenId];

pokemonName = pokemon.name;

sellingPrice = pokemonIndexToPrice[\_tokenId];

owner = pokemonIndexToOwner[\_tokenId];

}

function implementsERC721() public pure returns (bool) {

return true;

}

/// @dev Required for ERC-721 compliance.

function name() public pure returns (string) {

return NAME;

}

/// For querying owner of token

/// @param \_tokenId The tokenID for owner inquiry

/// @dev Required for ERC-721 compliance.

function ownerOf(uint256 \_tokenId)

public

view

returns (address owner)

{

owner = pokemonIndexToOwner[\_tokenId];

require(owner != address(0));

}

function payout(address \_to) public onlyCLevel {

\_payout(\_to);

}

// Allows someone to send ether and obtain the token

function purchase(uint256 \_tokenId) public payable {

address oldOwner = pokemonIndexToOwner[\_tokenId];

address newOwner = msg.sender;

uint256 sellingPrice = pokemonIndexToPrice[\_tokenId];

// Making sure token owner is not sending to self

require(oldOwner != newOwner);

// Safety check to prevent against an unexpected 0x0 default.

require(\_addressNotNull(newOwner));

// Making sure sent amount is greater than or equal to the sellingPrice

require(msg.value >= sellingPrice);

uint256 payment = uint256(SafeMath.div(SafeMath.mul(sellingPrice, 94), 100));

uint256 purchaseExcess = SafeMath.sub(msg.value, sellingPrice);

// Update prices

if (sellingPrice < firstStepLimit) {

// first stage

pokemonIndexToPrice[\_tokenId] = SafeMath.div(SafeMath.mul(sellingPrice, 200), 94);

} else if (sellingPrice < secondStepLimit) {

// second stage

pokemonIndexToPrice[\_tokenId] = SafeMath.div(SafeMath.mul(sellingPrice, 120), 94);

} else {

// third stage

pokemonIndexToPrice[\_tokenId] = SafeMath.div(SafeMath.mul(sellingPrice, 115), 94);

}

\_transfer(oldOwner, newOwner, \_tokenId);

// Pay previous tokenOwner if owner is not contract

if (oldOwner != address(this)) {

oldOwner.transfer(payment); //(1-0.06)

}

TokenSold(\_tokenId, sellingPrice, pokemonIndexToPrice[\_tokenId], oldOwner, newOwner, pokemons[\_tokenId].name);

msg.sender.transfer(purchaseExcess);

}

function priceOf(uint256 \_tokenId) public view returns (uint256 price) {

return pokemonIndexToPrice[\_tokenId];

}

/// @dev Assigns a new address to act as the CEO. Only available to the current CEO.

/// @param \_newCEO The address of the new CEO

function setCEO(address \_newCEO) public onlyCEO {

require(\_newCEO != address(0));

ceoAddress = \_newCEO;

}

/// @dev Assigns a new address to act as the COO. Only available to the current COO.

/// @param \_newCOO The address of the new COO

function setCOO(address \_newCOO) public onlyCEO {

require(\_newCOO != address(0));

cooAddress = \_newCOO;

}

/// @dev Required for ERC-721 compliance.

function symbol() public pure returns (string) {

return SYMBOL;

}

/// @notice Allow pre-approved user to take ownership of a token

/// @param \_tokenId The ID of the Token that can be transferred if this call succeeds.

/// @dev Required for ERC-721 compliance.

function takeOwnership(uint256 \_tokenId) public {

address newOwner = msg.sender;

address oldOwner = pokemonIndexToOwner[\_tokenId];

// Safety check to prevent against an unexpected 0x0 default.

require(\_addressNotNull(newOwner));

// Making sure transfer is approved

require(\_approved(newOwner, \_tokenId));

\_transfer(oldOwner, newOwner, \_tokenId);

}

/// @param \_owner The owner whose celebrity tokens we are interested in.

/// @dev This method MUST NEVER be called by smart contract code. First, it's fairly

/// expensive (it walks the entire Persons array looking for persons belonging to owner),

/// but it also returns a dynamic array, which is only supported for web3 calls, and

/// not contract-to-contract calls.

function tokensOfOwner(address \_owner) public view returns(uint256[] ownerTokens) {

uint256 tokenCount = balanceOf(\_owner);

if (tokenCount == 0) {

// Return an empty array

return new uint256[](0);

} else {

uint256[] memory result = new uint256[](tokenCount);

uint256 totalPokemons = totalSupply();

uint256 resultIndex = 0;

uint256 pokemonId;

for (pokemonId = 0; pokemonId <= totalPokemons; pokemonId++) {

if (pokemonIndexToOwner[pokemonId] == \_owner) {

result[resultIndex] = pokemonId;

resultIndex++;

}

}

return result;

}

}

/// For querying totalSupply of token

/// @dev Required for ERC-721 compliance.

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

return pokemons.length;

}

/// Owner initates the transfer of the token to another account

/// @param \_to The address for the token to be transferred to.

/// @param \_tokenId The ID of the Token that can be transferred if this call succeeds.

/// @dev Required for ERC-721 compliance.

function transfer(

address \_to,

uint256 \_tokenId

) public {

require(\_owns(msg.sender, \_tokenId));

require(\_addressNotNull(\_to));

\_transfer(msg.sender, \_to, \_tokenId);

}

/// Third-party initiates transfer of token from address \_from to address \_to

/// @param \_from The address for the token to be transferred from.

/// @param \_to The address for the token to be transferred to.

/// @param \_tokenId The ID of the Token that can be transferred if this call succeeds.

/// @dev Required for ERC-721 compliance.

function transferFrom(

address \_from,

address \_to,

uint256 \_tokenId

) public {

require(\_owns(\_from, \_tokenId));

require(\_approved(\_to, \_tokenId));

require(\_addressNotNull(\_to));

\_transfer(\_from, \_to, \_tokenId);

}

/\*\*\* PRIVATE FUNCTIONS \*\*\*/

/// Safety check on \_to address to prevent against an unexpected 0x0 default.

function \_addressNotNull(address \_to) private pure returns (bool) {

return \_to != address(0);

}

/// For checking approval of transfer for address \_to

function \_approved(address \_to, uint256 \_tokenId) private view returns (bool) {

return pokemonIndexToApproved[\_tokenId] == \_to;

}

/// For creating pokemon

function \_createPokemon(string \_name, address \_owner, uint256 \_price) private {

Pokemon memory \_pokemon = Pokemon({

name: \_name

});

uint256 newPokemonId = pokemons.push(\_pokemon) - 1;

// It's probably never going to happen, 4 billion tokens are A LOT, but

// let's just be 100% sure we never let this happen.

require(newPokemonId == uint256(uint32(newPokemonId)));

Birth(newPokemonId, \_name, \_owner);

pokemonIndexToPrice[newPokemonId] = \_price;

// This will assign ownership, and also emit the Transfer event as

// per ERC721 draft

\_transfer(address(0), \_owner, newPokemonId);

}

/// Check for token ownership

function \_owns(address claimant, uint256 \_tokenId) private view returns (bool) {

return claimant == pokemonIndexToOwner[\_tokenId];

}

/// For paying out balance on contract

function \_payout(address \_to) private {

if (\_to == address(0)) {

ceoAddress.transfer(this.balance);

} else {

\_to.transfer(this.balance);

}

}

/// @dev Assigns ownership of a specific Pokemon to an address.

function \_transfer(address \_from, address \_to, uint256 \_tokenId) private {

// Since the number of pokemons is capped to 2^32 we can't overflow this

ownershipTokenCount[\_to]++;

//transfer ownership

pokemonIndexToOwner[\_tokenId] = \_to;

// When creating new persons \_from is 0x0, but we can't account that address.

if (\_from != address(0)) {

ownershipTokenCount[\_from]--;

// clear any previously approved ownership exchange

delete pokemonIndexToApproved[\_tokenId];

}

// Emit the transfer event.

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

}

}

library SafeMath {

/\*\*

\* @dev Multiplies two numbers, throws on overflow.

\*/

function mul(uint256 a, uint256 b) internal pure returns (uint256) {

if (a == 0) {

return 0;

}

uint256 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 c;

}

/\*\*

\* @dev Substracts 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) {

uint256 c = a + b;

assert(c >= a);

return c;

}

}