pragma solidity ^ 0.5.1;

import "./Auction.sol";

import "./Strategy.sol";

/// @title Dutch Auction contract

/// @author Andrea Bruno 585457

/// @notice This contract implements the Dutch auction functionalities.

/// @dev The following comments are written using the Solidity NatSpec Format.

contract DutchAuction is Auction {

/// @dev This variable identifies the block number at the moment of contract deploy.

uint creationBlock;

/// @dev This variable identifies the block number of the winner bid.

uint winnerBlock;

/// @dev The reserve price decided by the seller.

uint reservePrice;

/// @dev The initial price decided by the seller.

uint initialPrice;

/// @dev The current price of the item.

uint actualPrice;

/// @dev To decrease the price a strategy is needed. Check the contract `Strategy` to learn more.

Strategy strategy;

/// @dev Those are the possible states of the auction.

enum State {

GracePeriod,

Active,

Validating,

Finished

}

State state;

/// @notice The constructor of the Dutch auction.

/// @dev The `msg.sender` will be the seller.

/// @param \_itemName is a short description of what is going to be sold.

/// @param \_reservePrice is the reserve price decided by the seller.

/// @param \_initialPrice is the initial price decided by the seller.

/// @param \_strategy the address of a `Strategy` contract.

constructor(

string memory \_itemName,

uint \_reservePrice,

uint \_initialPrice,

Strategy \_strategy

) public {

// Set into the description both the seller

description.seller = msg.sender;

// and the name of the item

description.itemName = \_itemName;

//Set state into "grace period"

state = State.GracePeriod;

reservePrice = \_reservePrice;

initialPrice = \_initialPrice;

actualPrice = \_initialPrice;

strategy = \_strategy;

//Remember the block number at the moment of contract deploy.

creationBlock = block.number;

}

/// @notice This function will activate the auction.

/// @dev Only the seller can invoke it.

function activateAuction() public onlySeller {

// In order to activate the auction we need to be in the "grace period"

require(state == State.GracePeriod, "To activate the contract you must be in the Grace Period");

// and also 5 minutes (20 blocks) must be elapsed.

require(block.number - creationBlock > 20, "Grace period is not finished yet");

// Set the current state to "Active".

state = State.Active;

// Set into the description the starting block of the auction.

description.startBlock = block.number;

// Communicate that the auction is started.

emit auctionStarted();

}

/// @notice This function will update the price.

/// @dev Since it is public, the gas will be spent by the people willing to know the current price.

/// @return The updated current price.

function getActualPrice() public returns(uint) {

// Compute the delta between the block in which the auction is started and the actual one.

uint deltaBlocks = description.startBlock - block.number;

// Let the strategy compute the price.

uint tmp = strategy.getPrice(actualPrice, -deltaBlocks);

// If the price computed is smaller than the reserve price

// set the actual price to the reserve price

// otherwise everything is fine.

if (tmp <= reservePrice) {

actualPrice = reservePrice;

} else {

actualPrice = tmp;

}

return actualPrice;

}

/// @notice Use this function to make a bid.

/// @dev The bidder should send a value greater or equal than the actual price.

/// @dev Finally, the auction passes in a "validation" state, to avoid inconsistencies due to forks on the blockchain

function bid() public payable {

// To maka a bid the auction need to be active.

require(state == State.Active, "This contract is not active yet");

// The bidder should send an appropriate value.

require(msg.value >= getActualPrice(), "The value sent is not sufficient");

description.winnerAddress = msg.sender;

description.winnerBid = msg.value;

winnerBlock = block.number;

//Activate the validation

validateAuction();

}

/// @dev This function allow to pass into a "validation" state.

function validateAuction() internal {

require(state == State.Active, "You can't validate a contract before activating it");

state = State.Validating;

}

/// @notice The following function finalize the contract and send the ether to the seller of the item.

/// @dev The security pattern Checks-Effects-Interaction is respected.

function finalize() public onlySeller {

require(state == State.Validating, "You can't finalize a contract before validation");

require(block.number - winnerBlock > 12, "For security reasons, you need to wait to validate the contract");

state = State.Finished;

// Emit the event on the blockchain

emit auctionFinished(description.winnerAddress, description.winnerBid, address(this).balance);

// Transfer the money to the seller

description.seller.transfer(description.winnerBid);

}

}