pragma solidity ^0.4.21;

contract SimpleAuction {

**// Parameters of the auction. Times are either**

**// absolute unix timestamps (seconds since 1970-01-01)**

**// or time periods in seconds.**

address public beneficiary;

uint public auctionEnd;

// Current state of the auction.

address public highestBidder;

uint public highestBid;

// Allowed withdrawals of previous bids

mapping(address => uint) pendingReturns;

// Set to true at the end, disallows any change

bool ended;

// Events that will be fired on changes.

event HighestBidIncreased(address bidder, uint amount);

event AuctionEnded(address winner, uint amount);

// The following is a so-called natspec comment,

// recognizable by the three slashes.

// It will be shown when the user is asked to

// confirm a transaction.

/// Create a simple auction with `\_biddingTime`

/// seconds bidding time on behalf of the

/// beneficiary address `\_beneficiary`.

function SimpleAuction(

uint \_biddingTime,

address \_beneficiary

) public {

beneficiary = \_beneficiary;

auctionEnd = now + \_biddingTime;

}

/// Bid on the auction with the value sent

/// together with this transaction.

/// The value will only be refunded if the

/// auction is not won.

function bid() public payable {

// No arguments are necessary, all

// information is already part of

// the transaction. The keyword payable

// is required for the function to

// be able to receive Ether.

// Revert the call if the bidding

// period is over.

require(now <= auctionEnd);

// If the bid is not higher, send the

// money back.

require(msg.value > highestBid);

if (highestBid != 0) {

// Sending back the money by simply using

// highestBidder.send(highestBid) is a security risk

// because it could execute an untrusted contract.

// It is always safer to let the recipients

// withdraw their money themselves.

pendingReturns[highestBidder] += highestBid;

}

highestBidder = msg.sender;

highestBid = msg.value;

emit HighestBidIncreased(msg.sender, msg.value);

}

/// Withdraw a bid that was overbid.

function withdraw() public returns (bool) {

uint amount = pendingReturns[msg.sender];

if (amount > 0) {

// It is important to set this to zero because the recipient

// can call this function again as part of the receiving call

// before `send` returns.

pendingReturns[msg.sender] = 0;

if (!msg.sender.send(amount)) {

// No need to call throw here, just reset the amount owing

pendingReturns[msg.sender] = amount;

return false;

}

}

return true;

}

/// End the auction and send the highest bid

/// to the beneficiary.

function auctionEnd() public {

// It is a good guideline to structure functions that interact

// with other contracts (i.e. they call functions or send Ether)

// into three phases:

// 1. checking conditions

// 2. performing actions (potentially changing conditions)

// 3. interacting with other contracts

// If these phases are mixed up, the other contract could call

// back into the current contract and modify the state or cause

// effects (ether payout) to be performed multiple times.

// If functions called internally include interaction with external

// contracts, they also have to be considered interaction with

// external contracts.

// 1. Conditions

require(now >= auctionEnd); // auction did not yet end

require(!ended); // this function has already been called

// 2. Effects

ended = true;

emit AuctionEnded(highestBidder, highestBid);

// 3. Interaction

beneficiary.transfer(highestBid);

}

}