pragma solidity ^0.4.24;

contract Bidding {

// Static bidding parameters.

address public owner;

string public city;

uint public distance;

uint public maxBidValue = 1000;

uint public winningBid= maxBidValue;

uint public biddingClose;

//uint public reputationfactor;

// Bidding will close wit hin specified seconds after the contract is deployed

address public winningBidder;

// Allowing withdrawals of previous bids

mapping(address => uint) returnsPending;

// Will be set true once the bidding is complete, preventing any further change

bool biddingComplete;

bool authorized;

struct Carrier {

bool authorized;

bool bidded;

uint reputationfactor;

//uint bidvalue; (not used right now)

}

mapping(address => Carrier) public carriers;

function authorize(address party, uint \_reputationfactor) ownerOnly public {

carriers[party].authorized = true;

carriers[party].reputationfactor= \_reputationfactor;

}

// Events to fire when change happens

event BidValueDecreased(address bidder, uint bidAmount);

event biddingResult(address winner, uint bidAmount);

event Failure(string \_message);

function Bidding(uint \_biddingTime, address \_beneficiary, string \_city, uint \_distance) {

owner = \_beneficiary;

biddingClose = now + \_biddingTime;

distance = \_distance;

city = \_city;

}

// Bidding is possible with the value sent along with this transaction

// Bid value is refunded if the bidding is not won

function bid() payable {

// The keyword payable is required so the function receives Ether.

require(now <= biddingClose,"Bidding is closed");

// If the bidding period is over, the call is reverted

require(msg.value < winningBid,"Bid amount is greater than maximum bid value");

// If the bid is not less, the money is sent back.

require(carriers[msg.sender].authorized,"Un-authorized carrier");

// If the carrier is not authorized, it cannot bid.

require(carriers[msg.sender].reputationfactor>5,"Carrier does not meet reputation factor criteria ");

carriers[msg.sender].bidded = true;

if (winningBidder != 0) {

returnsPending[winningBidder] += winningBid;

}

winningBidder = msg.sender;

winningBid = msg.value;

BidValueDecreased(msg.sender, msg.value);

}

// To withdraw a non-winning

function withdraw() returns (bool) {

require(now >= biddingClose,"Bidding process is still ungoing");

require (msg.sender != winningBidder,"You have won the bid, so cannot withdraw amount now.");

//address withdrawalaccount=msg.sender;

//uint withdrawalAmount = returnsPending[msg.sender];

uint bidAmount = returnsPending[msg.sender];

if (bidAmount > 0) {

returnsPending[msg.sender] = 0;

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

returnsPending[msg.sender] = bidAmount;

return false;

}

}

return true;

}

//When bidding ends, winning bid is sent to the beneficiary

function biddingClose() {

require (msg.sender == owner,"Not authorized to make this request");

// only owner can end the biddind

require(now >= biddingClose,"Bidding is not closed");

// bidding should not have ended

require(!biddingComplete,"Bidding in process");

// If this function has already been called, it should complete

biddingComplete = true;

biddingResult(winningBidder, winningBid);

// owner.transfer(winningBid);

}

function getbalance() public returns (uint){

require (msg.sender == owner);

return address(this).balance;

}

modifier ownerOnly() {

require(msg.sender == owner);

\_;

}

//modifier onlyEndedOrCanceled {

// if (now<biddingClose) throw;

// \_;

//}

}

***Updated program on 1 April 2019***

pragma solidity >=0.4.22 <0.6.0;

contract Biddingcontract {

// Static bidding parameters.

address public owner;

string public city;

uint public distance;

uint public maxBidValue = 1000;

uint public winningBid= maxBidValue;

uint public biddingClose;

//uint public reputationfactor;

// Bidding will close wit hin specified seconds after the contract is deployed

address public winningBidder;

// Allowing withdrawals of previous bids

mapping(address => uint) returnsPending;

// Will be set true once the bidding is complete, preventing any further change

bool biddingComplete;

bool authorized;

struct Carrier {

bool authorized;

bool bidded;

uint reputationfactor;

//uint bidvalue; (not used right now)

}

mapping(address => Carrier) public carriers;

function authorize(address party, uint \_reputationfactor) ownerOnly public {

carriers[party].authorized = true;

carriers[party].reputationfactor= \_reputationfactor;

}

// Events to fire when change happens

event BidValueDecreased(address bidder, uint bidAmount);

event biddingResult(address winner, uint bidAmount);

event Failure(string \_message);

function Bidding(uint \_biddingTime, address \_beneficiary, string memory \_city, uint \_distance) public{

owner = \_beneficiary;

biddingClose = now + \_biddingTime;

distance = \_distance;

city = \_city;

}

// Bidding is possible with the value sent along with this transaction

// Bid value is refunded if the bidding is not won

function bid() public payable {

// The keyword payable is required so the function receives Ether.

require(now <= biddingClose,"Bidding is closed");

// If the bidding period is over, the call is reverted

require(msg.value < winningBid,"Bid amount is greater than maximum bid value");

// If the bid is not less, the money is sent back.

require(carriers[msg.sender].authorized,"Un-authorized carrier");

// If the carrier is not authorized, it cannot bid.

require(carriers[msg.sender].reputationfactor>5,"Carrier does not meet reputation factor criteria ");

carriers[msg.sender].bidded = true;

if (winningBidder != address(0)) {

returnsPending[winningBidder] += winningBid;

}

winningBidder = msg.sender;

winningBid = msg.value;

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

}

// To withdraw a non-winning

function withdraw() public returns (bool) {

require(now >= biddingClose,"Bidding process is still ungoing");

require (msg.sender != winningBidder,"You have won the bid, so cannot withdraw amount now.");

//address withdrawalaccount=msg.sender;

//uint withdrawalAmount = returnsPending[msg.sender];

uint bidAmount = returnsPending[msg.sender];

if (bidAmount > 0) {

returnsPending[msg.sender] = 0;

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

returnsPending[msg.sender] = bidAmount;

return false;

}

}

return true;

}

//When bidding ends, winning bid is sent to the beneficiary

function biddingClosefun() public {

require (msg.sender == owner,"Not authorized to make this request");

// only owner can end the biddind

require(now >= biddingClose,"Bidding is not closed");

// bidding should not have ended

require(!biddingComplete,"Bidding in process");

// If this function has already been called, it should complete

biddingComplete = true;

emit biddingResult(winningBidder, winningBid);

// owner.transfer(winningBid);

}

function getbalance() public returns (uint){

require (msg.sender == owner);

return address(this).balance;

}

modifier ownerOnly() {

require(msg.sender == owner);

\_;

}

//modifier onlyEndedOrCanceled {

// if (now<biddingClose) throw;

// \_;

//}

}