pragma solidity ^0.4.24;

contract cargo\_shipping{

uint public value;

//seller details

struct seller

{

string sname;

string saddress;

}

//cargo info

struct Cargo{

string name;

string description;

uint hscode;

uint quantity;

uint weight;

//string origin;

string destination;

uint startdate;

uint deadline;

uint penalty;

uint hash;

}

//ship info

struct Ship{

string vessel;

string voyage;

//bool active;

}

seller S1;

//Cargo public cargo;

Ship public ship;

address public Seller;

address public Customer;

address public shipper;

Cargo public setCargo;

function cargo\_shipping(string \_sname,

string \_saddress,

string cargoname,

string \_description,

uint \_hscode,

uint \_quantity,

uint \_weightinkg,

uint \_hash,

string \_origin,

string \_destination,

//uint \_penalty,

uint \_deadline

// string \_vessel,

// string \_voyage

)public payable{

S1.sname = \_sname;

S1.saddress = \_saddress;

Seller = msg.sender;

value=msg.value/2;

require((2\*value) == msg.value);

setCargo.name = cargoname;

setCargo.description = \_description;

setCargo.hscode= \_hscode;

setCargo.quantity = \_quantity;

setCargo.weight = \_weightinkg;

setCargo.hash= \_hash;

//setCargo.origin = \_origin;

setCargo.destination = \_destination;

//setCargo.penalty= \_penalty;

setCargo.deadline = \_deadline;

//ship.vessel = \_vessel;

//ship.voyage = \_voyage;

}

enum State { Created, Locked, Inactive }

State public state;

modifier condition(bool \_condition) {

require(\_condition);

\_;

}

modifier onlyCustomer() {

require(msg.sender == Customer);

\_;

}

modifier onlySeller() {

require(msg.sender == Seller);

\_;

}

modifier inState(State \_state) {

require(state == \_state);

\_;

}

event Aborted();

event PurchaseConfirmed();

event Confirmshipping();

event ItemReceived();

event delayedShipment(string s,uint amount);

/// Abort the purchase and reclaim the ether.

/// Can only be called by the seller before

/// the contract is locked.

function abort()

public payable

onlySeller

inState(State.Created)

{

emit Aborted();

state = State.Inactive;

Seller.transfer (address(this).balance);

}

function Confirmshipper(string \_vessel,string \_voyage)

public

condition(msg.value == (2\* value))

payable

{

//.name = cargoname;

ship.vessel = \_vessel;

ship.voyage = \_voyage;

emit Confirmshipping();

shipper =msg.sender;

}

/// Confirm the purchase as buyer.

/// Transaction has to include `2 \* value` ether.

/// The ether will be locked until confirmReceived

/// is called.

function confirmPurchase()

public

inState(State.Created)

condition(msg.value == (2\* value))

payable

{

emit PurchaseConfirmed();

Customer = msg.sender;

setCargo.startdate= now;

state = State.Locked;

}

/// Confirm that you (the buyer) received the item.

/// This will release the locked ether.

function confirmReceived()

public

onlyCustomer

inState(State.Locked)

{

emit ItemReceived();

// It is important to change the state first because

// otherwise, the contracts called using `send` below

// can call in again here.

state = State.Inactive;

uint arrivaldate=now;

uint delay=arrivaldate-setCargo.startdate;

if (delay> setCargo.deadline)

{

Customer.transfer(2\*value);

shipper.transfer(value);

Seller.transfer(address(this).balance);}

else

// NOTE: This actually allows both the buyer and the seller to

// block the refund - the withdraw pattern should be used.

{Customer.transfer(value);

shipper.transfer(2\*value);

Seller.transfer(address(this).balance);}

}

}