Course 2 Project: Design and Development of Smart Contract

Title: Auction for Fundraising for a Social Cause

Learning objectives:

- To design, develop and test a smart contract for a problem using Solidity language and Remix IDE. (Module 1, and Module 2)
- To apply incremental development method and best practices discussed in the course. (Module 3 and Module 4)

Problem Statement:

Consider the problem of Chinese auction or penny social. We will refer to it as simple "Auction." It is a conventional approach used for fundraising for a cause. The organizers collect items to be auctioned off for raising funds. Before the auction, the items for auctions are received and arranged each with a bowl to place the bid. A chairperson is a special person among the organizers. She/he heads the effort and is the only person who can determine the winner by random drawing at the end of the auction. A set of bidders buy sheets of tickets with their money. The bidder's sheet has a stub that identifies the bidder's number, and tokens bought.

The bidders examine the items to bid, place the one or more tickets in the bowl in front of the items they desire to bid for until all the tickets are used. After the auction period ends the chairperson, collects the bowls, randomly selects a ticket from each item's bowl to determine the winning bidder for that item. The item is transferred to the winning bidder. Total money collected is the fund raised by the penny social auction.

Assumptions:

The description given above is for a general penny social auction. For the sake of our project implementation we will introduce some simplifying assumptions. Here they are:

1. Fixed number of bidders, initialized to 4. All 4 need to self-register. Funds transfer from bidder is automatically done and is not in the scope of this application.
2. Fixed number of items to be auctioned off, initialized to 3.
3. Items auctioned are indexed from 0..N-1 where N is the number of items for auction. N is 2.
4. Each bidder buys just 1 sheet of tickets or tokens; each sheet has only 5 tokens.
5. Assume simple number for the serial numbers for the sheet of tickets: 0,1,2,3. Here we show the tokens of bidder 0 and 1.

0	0	0	0	0
1	1	1	1	1
.	.	.	.	.
n-1	n-1	n-1	n-1	n-1

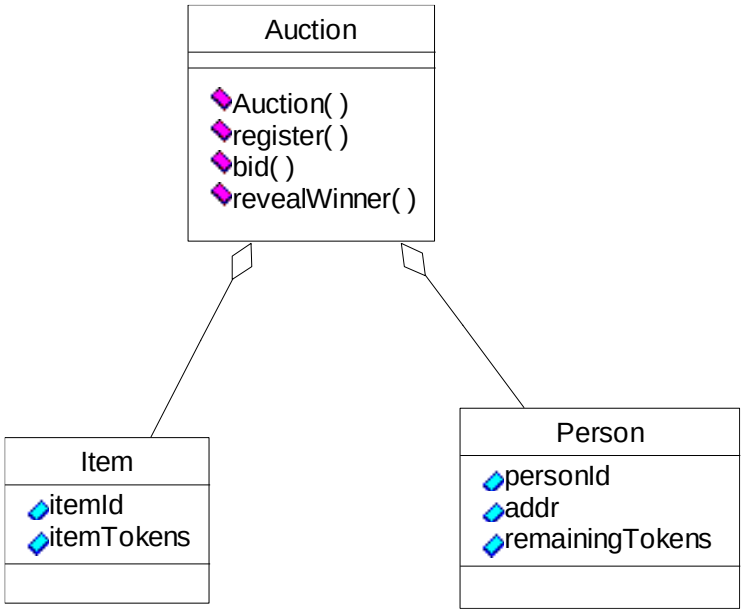
For n people and so on.

The Design:

Let's design the smart contract for this. Visualize the situation using the screen shot given below.

These are the pictures of items, we need to define "Items" in the smart contract. We will also need to define Persons or bidders who will bidding on the Items. We will also need some supporting data variables.

The functions are similar to the Ballot of our lessons. Constructor that initializes the owner, register that allows (decentralized person) to register online to get the tokens and start bidding, bid function that lets a person bid, and finally revealWinner, to randomly choose the winner for the item auctioned. Here is our design. Always remember to design first. The Auction smart contract has Item and Person structs and other data items such as array of Items, array of Persons, array of winners, mappings, and beneficiary address.



Implementation:

You will write two versions of the implementation templates: version 1: without modifier (80%) and version 2 with modifier (20%). We have provided the templates for both versions.

1. Please understand the problems before you proceed.
2. Copy the template into your Remix IDE. Complete the code. You will need to fill in the code at *ONLY* at the locations indicated. Test it to see if it is operational.
3. Implement version of as Auction.sol and submit for grading. Then update this base version for Auction.sol for Part 2 requirements, and submit for grading.

Version 1: (80%) Auction.sol: There are 6 tasks where you will fill in the code. You need to review the code given and understand it fully before you start adding code. This partial code is available in the course resources section as Auction.sol.

Testing on Remix:

Test the completed code by compiling and running it on Remix JavaScript VM. (i) Account 0 provided by JavaScript VM is the Auction beneficiary and will not bid. (ii) Navigate to each of the other accounts, and "register." You will register four bidders using register function. (iii) The next step is for the bidders to each bid; for test purposes, you can execute the "bid" function with {{0,1}{1,1}{2,1} } for each of the four accounts. (iv) Next, execute the "revealWinner" function to determine the winner for each item randomly, and (v) the getter of the "winners" data can be executed with {0, 1 and 2} as a parameter in sequence to reveal the winners of the draw respectively. You can do a lot of more testing and exploration with the buttons provided by the Remix web interface.

Partial code for Auction.sol

```
pragma solidity ^0.4.17;
contract Auction {

// Data
    //Structure to hold details of the item
    struct Item {
        uint itemId; // id of the item
        uint[] itemTokens;  //tokens bid in favor of the item

    }

   //Structure to hold the details of a persons
    struct Person {
        uint remainingTokens; // tokens remaining with bidder
        uint personId; // it serves as tokenId as well
        address addr;//address of the bidder
    }

    mapping(address => Person) tokenDetails; //address to person
    Person [4] bidders;//Array containing 4 person objects

    Item [3] public items;//Array containing 3 item objects
    address[3] public winners;//Array for address of winners
    address public beneficiary;//owner of the smart contract
```

```
    uint bidderCount=0;//counter

    //functions

    function Auction() public payable{    //constructor

    //Part 1 Task 1. Initialize beneficiary with address of smart contract's
owner
    //Hint. In the constructor,"msg.sender" is the address of the owner.
            // ** Start code here. 1 line approximately. **/

            //** End code here. **/
        uint[] memory emptyArray;
        items[0] = Item({itemId:0,itemTokens:emptyArray});

        //Part 1 Task 2. Initialize two items with at index 1 and 2.
            // ** Start code here. 2 lines approximately. **/
        items[1] =
        items[2] =
            //** End code here**/
    }


    function register() public payable{


        bidders[bidderCount].personId = bidderCount;

        //Part 1 Task 3. Initialize the address of the bidder
        /*Hint. Here the bidders[bidderCount].addr should be initialized with
address of the registrant.*/

            // ** Start code here. 1 line approximately. **/

            //** End code here. **

        bidders[bidderCount].remainingTokens = 5; // only 5 tokens
        tokenDetails[msg.sender]=bidders[bidderCount];
        bidderCount++;
    }

    function bid(uint _itemId, uint _count) public payable{
        /*
            Bids tokens to a particular item.
            Arguments:
            _itemId -- uint, id of the item
            _count -- uint, count of tokens to bid for the item
        */

        /*
            Part 1 Task 4. Implement the three conditions below.
                  4.1 If the number of tokens remaining with the bidder is <
count of tokens bid, revert
                  4.2 If there are no tokens remaining with the bidder,
            revert.
```

```
                4.3 If the id of the item for which bid is placed, is
greater than 2, revert.

            Hint: "tokenDetails[msg.sender].remainingTokens" gives the
details of the number of tokens remaining with the bidder.
            */

        // ** Start code here. 2 lines approximately. **/


        //** End code here. **

            /*Part 1 Task 5. Decrement the remainingTokens by the number of
tokens bid
            Hint. "tokenDetails[msg.sender].remainingTokens" should be
decremented by "_count". */

            // ** Start code here. 1 line approximately. **


            //** End code here. **
        bidders[tokenDetails[msg.sender].personId].remainingTokens=
tokenDetails[msg.sender].remainingTokens; //updating the same balance in
bidders map.
        Item storage bidItem = items[_itemId];
        for(uint i=0; i<_count;i++) {
            bidItem.itemTokens.push(tokenDetails[msg.sender].personId);
        }
    }


    function revealWinners() public {

            /*
                  Iterate over all the items present in the auction.
                  If at least on person has placed a bid, randomly select
the winner */

        for (uint id = 0; id < 3; id++) {
            Item storage currentItem=items[id];
            if(currentItem.itemTokens.length != 0){
       // generate random# from block number
            uint randomIndex = (block.number /
currentItem.itemTokens.length)% currentItem.itemTokens.length;
// Obtain the winning tokenId

                uint winnerId = currentItem.itemTokens[randomIndex];

/* Part 1 Task 6. Assign the winners.
Hint." bidders[winnerId] " will give you the person object with the winnerId.
you need to assign the address of the person obtained above to winners[id] */

                        // ** Start coding here *** 1 line approximately.


                        //** end code here*
```

```
            }
        }
    }

  //Miscellaneous methods: Below methods are used to assist Grading. Please
DONOT CHANGE THEM.
 function getPersonDetails(uint id) public constant
returns(uint,uint,address){
      return
(bidders[id].remainingTokens,bidders[id].personId,bidders[id].addr);
}

}
```

Version 2: (20%) After testing version 1 on Remix IDE, please submit the solution for autograding on Coursera.

For version 2, we need to add a modifier so that only the owner can invoke the function “revealWinner”.

```
/*Part 2 Task 1. Create a modifier named "onlyOwner" to ensure that only
owner is allowed to reveal winners.

Hint: Use require to validate if "msg.sender" is equal to the
"beneficiary". */

This involves defining a modifier and using the modifier in the header of
the “revealWinner” function.

Add these updates, compile and test with Remix IDE. Make sure the
operations of “revealWinner” from any other account than the Owner
(account{0}) results in “revert”.

Save the completed solution and submit for auto grading.

END
```