Desafios Petshop Solidity

Aluno: Marcos Paulo Ferreira

Junto com este documento está o arquivo Adoption.sol com todo o código.

Desafio 1

A função de doação agora possui um parâmetro de "valor" que é o quanto um comprador está disposto a pagar pelo Pet. Somente os Pets em estado "A_VENDA" é que podem ser comprados.

```
/* Funcao de adocao. Aceita o id do pet e o valor */
  function adopt(uint petId, uint value)
        validPet(petId)
        public
        payable
        returns(address, uint)
        {
           require(value == msg.value);
           Pet storage pet = pets[petId];

           if(pet._status == STATUSPET.A_VENDA) {
                Comprador memory c = Comprador(msg.sender, msg.value) ;

                c.endereco = msg.sender;
                c.valor = msg.value;

                compras[petId].push(c);

                 return (c.endereco, c.valor);
                 } else { /* Pet ja vendido mas recebeu proposta de compra */
                    emit PetJaComDono(petId, pets[petId].adopter, msg.sender) ;
                 }

                 /* retorno para fins de Debug */
                 return (address(0), 0);
          }
```

A função garante que o valor do parâmetro seja igual ao valor enviado na transação. Ao adotar um Pet, o comprado está na verdade dando um "lance" e não concretizando a compra. A compra só é efetivada pelo dono do contrato.

Cada pedido de adoção é empilhado em um mapping chamado "compras".

Desafio 2

A função aceitaCompraPet() concretiza a venda do Pet. Ela só pode ser executada pelo dono do contrato.

Basicamente, o dono passa como parâmetro o id do Pet e também a posição da compra a ser aceita.

A função concretiza a compra do parâmetro e reembolsa para os compradores rejeitados o valor enviado.

```
/* Chamado pelo dono do contrato. Finaliza a compra de um Pet */
    function aceitaCompraPet(uint petId, uint index)
    validCompra(petId, index)
    public
    {
       require(owner == msg.sender);

       address endereco;
       uint valor;

       /* Remove a compra e devolve o dinheiro */
       for(uint i=0; i < compras[petId].length; i++)
       {
          if(i != index ) {
             endereco = compras[petId][i].endereco;
             valor    = compras[petId][i].valor;

              emit Reembolso(endereco, valor);
             /* TODO: Verificar metodo melhor pra evitar reentrancy */
             if(valor > 0)
             {
                endereco.transfer(valor);
                /* Evita reenvio por reentrada */
                compras[petId][i].valor = 0;
             }
          }
       }

       pets[petId]._status = STATUSPET.VENDIDO;
       pets[petId].adopter = compras[petId][index].endereco;
       pets[petId].preco = compras[petId][index].valor;

       emit PetComprado(index, pets[petId].name, pets[petId].adopter, pets[petId].preco) ;

       delete compras[petId];
    }
```

O Pet vendido muda para o estado VENDIDO e não poderá ser mais comprado.

Demais funções também implementadas:

```
    function getTotalComprasByPet(uint petId);
```

Obtém total de pedidos de para um Pet.

```
    function getCompraPet(uint petId, uint index);
```

Retorna uma compra (endereço + valor) de determinado Pet.

function getCompraMaisCaraPet(uint petId)

Retorna o pedido de compra mais caro (de maior valor) de determinado Pet. O dono do contrato pode usar essa função para decidir qual comprador levará o Pet.

Desafio 3

Os logs implementados foram:

event PetComprado(uint idCompra, string nome, address comprador, uint valor);

é chamado toda vez que o dono do contrato efetiva a venda de um Pet.

event PetJaComDono(uint petId, address comprador_original, address comprador_negado);

é chamado quando alguém tenta comprar um Pet que já tem dono. A compra é recusada e o evento é emitido.

event Reembolso(address comprador, uint valor);

é chamado quando os compradores rejeitados recebem o reembolso do pedido.