Boson Protocol

Engineer Technical Interview - Problem Statement

Context

Boson Protocol is advancing the world of commerce by enabling enterprises, organisations, and customers to bridge the divide between digital decentralized technologies and the transfer and trade of physical goods.

Our vision is for Boson to become the basic plumbing for dCommerce and its data on the emerging decentralized web, where the value captured is distributed equitably between token holders and protected from capture by a single centralized entity. Read more about us here: https://bosonprotocol.io/

Guidelines

- The following exercise contains some example input and expected outputs, however the solution should be extendable to similar inputs and outputs.
- You can solve the problem preferably in Solidity or the language of your choice, although we might not be familiar with it, so be prepared to explain it!
- The exercise is designed to understand how you approach problem solving. If your specialism doesn't include blockchain, you don't have to use smart contracts.
- Either submit the solution as an email attachment (.zip) or use something like Remix and share via a Github gist.
- What we will look for:
    - a good breakdown of the core domain,
    - a simple interface, we don't expect UIs, web APIs or database integration
    - a flexible solution, that can be extended in various different ways,
    - a well tested solution,
    - a simple solution, however a library shouldn't solve the problem for you
    - clear instructions on how to run the solution and tests,
    - a brief explanation of your design and assumptions,
- We want our hiring practices to be fair. To ensure this, please do not publically share the problem or solution.

Problem

At its core, Boson Protocol makes use of an escrow arrangement to securely hold funds until a commercial transaction has completed. In this exercise, we explore how escrow works in a naïve version of the Boson Protocol core exchange mechanism.

Our system involves two parties, *Sellers* and *Buyers*. Each party has an *Account* with a balance (in whatever currency you like), which they can *Credit* with funds and from which they can make payments.

The parties engage in the following activities, respectively:

- *Sellers*:
    - *Offer* items for sale, with each item consisting of a title and a price; and
    - Fulfil orders for items, performed out-of-band.
- *Buyers*:
    - *Order* items offered by sellers;
    - *Complete* orders once they have received their purchase; and
    - *Complain* about orders if they never receive their purchase.

The escrow arrangement functions as follows:

- When a *Buyer* places an *Order* for an item, they place payment into escrow for that item.
- When a *Buyer* receives their purchase and *Completes*, the payment is paid from escrow to the *Seller.*
- When a *Buyer* fails to receive their purchase and *Complains*, the payment is refunded from escrow to the *Buyer*.
- If the *Buyer* does nothing, the payment remains in escrow.

Your goal is to build a system to process such transactions according to the example input provided below.

Example input:

Note: you don't need to parse this text in this format. Feel free to convert to another format if needed. However, it should be easy to run your solution with different inputs.

```
---
Buyer 1  | Credit   | 20
Buyer 2  | Credit   | 40
Seller 1 | Offer    | Coffee, 3
Seller 2 | Offer    | T-Shirt, 5
Seller 1 | Offer    | Tea, 2.5
Seller 1 | Offer    | Cake, 3.5
Seller 2 | Offer    | Shorts, 8
Seller 2 | Offer    | Hoody, 12
Buyer 1  | Order    | T-Shirt
Buyer 1  | Credit   | 10
Buyer 2  | Order    | Hoody
Buyer 1  | Complete | T-Shirt
Buyer 1  | Order    | Coffee
Buyer 1  | Order    | Cake
Buyer 2  | Complain | Hoody
Buyer 2  | Order    | Tea
Buyer 1  | Complete | Coffee
```

Questions:

Using the above information, please make sure your solution is able to answer the following:

1. What is *Buyer 1*'s balance?
2. What is *Seller 2*'s balance?
3. What is the total amount held in escrow?

Example output:

```
1. 18.5
2. 5
3. 6
```