Deposit Plan Case

Background Information

At StashAway, our customers can create one or more portfolios to invest for different purposes (for example: a retirement portfolio and an emergency funds portfolio).

Customers deposit funds into StashAway via bank transfer and have to include their reference code when making the transfer. This reference is unique per customer, and therefore does not include any information into which portfolios the funds should be allocated.

As such, customers are encouraged to set up a deposit plan to specify which portfolios incoming funds should be allocated to. The customer can set up multiple deposit plans on either a one-time and/or monthly basis. The monthly plan exists for convenience, so that customers can set up a recurring deposit and don't have to login regularly to specify the split.

Please feel free to explore either our FAQs or our product in case you have questions about particular aspects of deposits at StashAway. You don't have to replicate StashAway's logic, as as long as the solution you build is reasonable for the customer.

An example

A customer has 2 portfolios named:

- "High risk"
- "Retirement"

As well as 2 deposit plans:

- One-time (High risk: $10,000, Retirement: $500)
- Monthly (High risk: $0, Retirement: $100)

The customer made 2 deposits:

- First deposit: $10,500
- Second deposit: $100

The result of the fund deposit allocation should be:

- High risk portfolio gets 10,000$
- Retirement portfolio gets 600$

Keep in mind that this is a happy case, and there are other edge cases you should handle.

Task

Build a function that takes in the following input:

- A list of 1 one-time and/or 1 monthly deposit plan (max 2 plans total)
- A list of deposits for a customer

This function must return the final allocation of funds among the customer's portfolios (see above example).

When you design the data structures, keep in mind:

- Each deposit plan must contain the related portfolios for that plan & the absolute amount of money to allocate to each portfolio.
- Deposits can only contain the absolute amount of money & a reference code. Deposits cannot have any deposit plan linked to them - the bank processing customer transfers doesn't know or care about deposit plans.

All deposits must be distributed fully, i.e. please leave no leftover money. If during implementation, you identify other non-happy scenarios, don't just throw an error - try to consider a more elegant approach.

💡We assess a number of things including how you understand the problem statements, your solution's technical design, and your ability to validate the correctness of your solution (**hint: unit tests are your friend**). While this is a small problem, we expect you to submit what you believe is **production-quality code**: code that you can maintain & evolve when new requirements come in.

Good luck!

Submission

You may deploy your solution using **Github** in either one of the following languages: Typescript, Node.JS, Python, Java, Kotlin, Go.

Once the assignment is deployed, please make sure the repository is private. You can take it down after we submit our feedback.

You can then upload the Zip file on Kula.

More questions?

Q: Should I use RDBMS/SQL or NoSQL? How should I store deposit plan, portfolio data?
A: A DB layer is not mandatory. If you still want to include a DB layer, the choice is up to you!

Q: Do I need to integrate a UI? Is the expected output a website that takes in a CSV or manual entry?
A: The base requirement is just a function, which can be called via CLI or test suite. If you'd like to provide a UI or other method of input, feel free to do so, but keep in mind that we are most concerned with your business logic for allocating deposits, so please make sure you have a solid solution there.

Q: Does a monthly deposit plan have no high risk? Or that is something that is configurable?
A: Deposit plans are configurable by the customer.

Q: Should I distribute the funds using <approach A>, <approach B>, etc?
A: The distribution logic is for you to propose, you can think through the problem or do some research. What's important is that: it is logical & consistently applied, it handles edge cases, and it ensures full distribution of funds (including cents).