Interview Assignment

Instructions

Following are sample scenarios for a system we would like you to design. Please pick one (or a reasonably-sized subset) and implement it in a test-driven fashion using Kotlin.

While developing a solution, please keep in mind non-functional requirements for the whole system and the other scenarios to the extent that it should avail for them (at least in theory, which you should be able to argue for)

Your solution should include a readme file with:

- instructions for building and running the tests (your “demo”)
- any comments you think might help us understand your design/thoughts

Problem Domain

Our system provides a set of APIs for selling and servicing annuity products. Anyone can apply for an annuity policy, and if the application is approved, they pay the premium and enjoy the policy. The policy pays guaranteed interest over a fixed term and, depending on the type, can provide a monthly stream of payments. After the end of the term, the owner receives the balance plus interest. It also serves as a life insurance: in the event of the owner’s death the money is paid to the designated beneficiaries.

Non-functional Requirements

- High availability
- Horizontal scalability: we want to be able to support traffic surges by dynamically scaling our system out
- Transactional consistency where we need it, eventual consistency with tunable SLAs otherwise
- Privacy: no unencrypted personally-identifying information about our customers can ever be exposed, even in case of a security breach
- Auditing: as this is a highly-regulated financial service, we need to be able to provide audit records for every event of importance that happened in the system for the last year

Scenarios

Customer applies for a policy

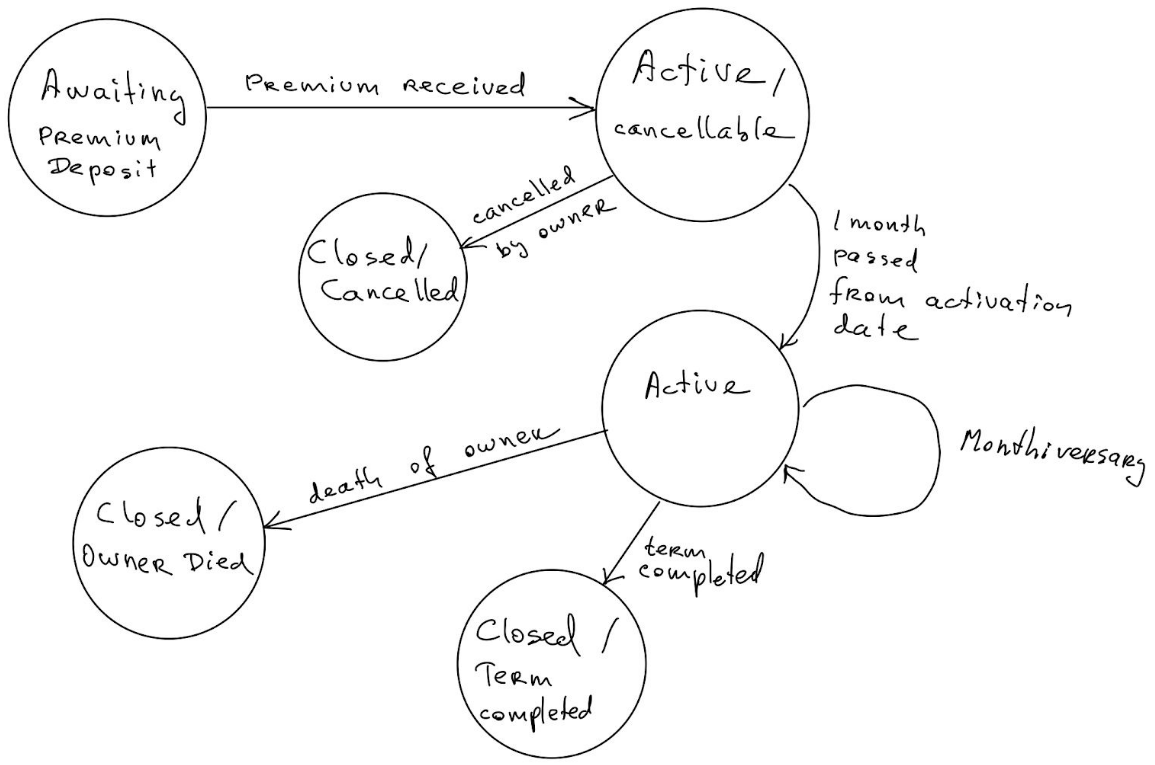
Having chosen a product, the customer fills out an annuity policy application. The application goes through a verification workflow that answers questions like these:

- Does the bank account provided by the applicant exist?
- Does it have enough funds for the premium deposit?
- Is the applicant a known money launderer?

The verification may take a couple of days. If everything checks out, the policy application is approved and we are ready to create a policy.
Implementation note: for answering the questions above, let's use plaid.com to check bank info and account balance (it wants a callback endpoint to give you the info asynchronously, so just implement the endpoint), and the fictional bigbrother.com to verify the applicant is not a known criminal - let's pretend it offers a jar with Java API, so just invent its contract and stub it out.

Policy life cycle

Once we know the application has been accepted, we can create a policy for the customer. This diagram illustrates the state transitions of the policy over its lifetime, each explained below.



After the application is approved, a new policy is created in *Awaiting Premium Deposit* state. Upon entering this state, it requests its premium deposit.
As soon as deposit is received, it transitions to *Active/Cancellable*, where it stays for 1 month. During this time the owner can cancel it and get the whole premium back if he changes his mind. If the owner does cancel it within a month, the policy becomes *Closed/Cancelled*.
After a month, the policy transitions to *Active*, after which point the owner will have to pay a fee to get his premium back.
The monthly anniversary of a policy is called its *monthiversary*. On this day, monthly interest is calculated and payout transactions are requested (a *payout* is the money paid to the owner every month). Also, this is when the policy transitions to *Closed/Term Completed* if its term has run out, and the request to pay the remaining balance (premium + interest - payouts up to date) to the owner is created.
If the owner of an active policy dies, it transitions to *Closed/Owner Died*, and transactions to pay the money to the beneficiaries are generated.

Reporting

All financial transactions generated by the policies, such as premium deposits, payouts or interest payments must go through a legacy system maintained by another team. We integrate with them by exposing a set of REST APIs that they can query. Different reports are needed at different times:

- Daily transaction report, used to initiate ACH transactions
- Yearly tax report by policy, used to issue tax forms to the customers
- Monthly withholding report, used to withhold taxes on amounts paid