Proyecto 1 - Pumabank

Integrantes

- 321671958 - *Cisneros Álvarez Danjiro*.
- 422083399 - *Teniente Ornelas Oscar Manuel*.
- 422109167 - *Tenorio Reyes Ihebel Luro*.

Descripcion

Nuestro programa ataca el problema de forma que puede entenderse como la terminal de un banco, donde cualquier persona puede llegar y comenzar a ejecutar operaciones en él una vez que entra con las credenciales correctas.

Instrucciones

Compilación y ejecución

Con **Docker**, se puede compilar y ejecutar
Compilar:
`docker build -t pumabank-app .`
Ejecutar:
`docker run -it -v ./datos-persistentes:/app/generados pumabank-app`
En esta forma existen problemas con los txt persistentes.

> ⚠ **Nota**
>
> En algunas distribuciones hay que ejecutar ambos comandos con sudo

Con **Java** (y con comandos de linux):
Compilar:
`mkdir -p bin && javac -d bin src/PUMABANK/**/*.java`
Ejecutar:
`java -cp bin PUMABANK.PumaBankApp`

Cuentas disponibles:

Cuenta	Nip
PUMALACTICO 001	1234
SUPER_NOVA 002	2468
GALAXIA_M31 003	1357
ANDROMEDA 004	0000
NEBULA_DE_ORION 005	9999
PORT_C1	7777
PORT_C2	8888

Patrones de diseño

State

State fue el primer patrón más claro, y es que los estados de cuenta cuentan (por composición) un estado el cual representa la distintas formas en que puede estar, activa, sobregirada, cancelada, etc.

Proxy

Proxy nos permite interactuar con cada cuenta pero primero asegurándose de que el usuario se autentifique de forma correcta y luego como intermediario interactuar con la cuenta real.

Strategy

Strategy permite que cada cuenta pueda ejecutar sus intereses según sean mensuales, anuales o premium.

Iterator y Composite

Decidimos colocarlos juntos porque el programa funciona de tal forma que tenemos un Registro de todas las cuentas que existen en el banco por lo que iterate nos permite ejecutar funciones en todas ellas como update(), además de esto como los portafolios en esencia son carpetas de más cuentas, decidimos que implementaría la misma interfaz que una cuenta, haciéndolo entonces delegar todo su trabajo a sus hojas (las cuentas), por lo que se implementa composite (y también iterator).

Decorator

Decorator nos permite añadir los servicios extras a cada cuenta, realmente no hay mucho más que decir en esta parte.

Observer

Por último, observer nos permite que cada cuenta haga el respectivo cobro de sus servicios y reciba el pago de sus tipos de interés mensualmente con la función update().