README – PROYECTO 1

PROYECTO 1 - PumaBank

Equipo: LasPamparas Integrantes:

- Aviles Luque Luis Diego - 318176653
- Luis Enrique Flores Juárez - 322278316
- Julio César Islas Espino - 32034 0594

Proyecto 1 - Modelado y Programación

PumaBank es un **banco digital** capaz de manejar cuentas con políticas de cálculo de interés intercambiables, adaptarse automáticamente al estado actual de cada cliente y notificar en tiempo real sobre cualquier evento importante..

Los objetivos concretos son:

- Demostrar el uso de patrones (Strategy, State, Observer, Composite, Decorator, Factory, Proxy) en un sistema cohesionado.
- Mantener reglas de negocio claros y verificables (sobregiro, congelada, reabrir cuentas, cargos, NIP).
- Facilitar la extensión del sistema (nuevos tipos de intereses, servicios adicionales, más estados).
- Permitir pruebas y ejecuciones simples (ej.: proceso mensual automatizado).

Argumentación sobre la aplicación de los patrones

Para la solución de este proyecto se implementaron 8 patrones de diseño clásicos con el objetivo de construir un sistema bancario robusto, flexible y mantenible, adhiriéndose a los principios SOLID. A continuación se argumenta la elección de cada uno:

1. ***State***

• Razón de uso:* Se eligió para modelar los diferentes estados de una cuenta bancaria (Activa , Sobregirada , Congelada , Cerrada). Permite que el objeto Cuenta altere su comportamiento dinámicamente cuando su estado interno cambia (e.g., no permitir retiros si

está congelada) sin usar condicionales complejos, encapsulando la lógica de cada estado en su propia clase.

2. ***Strategy***

• Razón de uso: Se utilizó para definir una familia de algoritmos intercambiables para el cálculo de intereses (InteresMensual , InteresAnual , InteresPremium). Esto permite cambiar la política de intereses de una cuenta en tiempo de ejecución sin modificar la clase Cuenta , promoviendo el principio de Abierto/Cerrado.

3. ***Observer***

• Razón de uso: Se implementó para notificar a múltiples clientes (observadores) sobre eventos relevantes en sus cuentas (sujeto), como balances bajos o cargos inesperados. Desacopla al emisor de las notificaciones (GestorAlertas) de los receptores (ClienteObservador), permitiendo añadir nuevos tipos de notificaciones o suscriptores fácilmente.

4. Decorator

• Razón de uso: Se aplicó para añadir funcionalidades y servicios adicionales a una cuenta (SeguroAntifraude , Recompensas) de forma dinámica y acumulativa. Evita la explosion de subclases que ocurriría si se intentara modelar todas las combinaciones de servicios, permitiendo "envolver" un objeto base con nuevas responsabilidades.

5. ***Proxy***

• Razón de uso: Se empleó para controlar el acceso a las operaciones bancarias. El AccesoRemoto actúa como un proxy que verifica el NIP del usuario y controla los intentos fallidos antes de delegar la llamada al objeto real (OperacionesBancariasImpl), añadiendo una capa de seguridad de forma transparente.

6. ***Factory Method***

• Razón de uso: Se utilizó para centralizar y simplificar la creación de distintos tipos de cuentas. FabricaCuentas proporciona metodos (crearCuentaMensual , crearCuentaPremium) que encapsulan la lógica de instanciación y configuracion de cada tipo de cuenta, asegurando que los objetos se creen en un estado consistente.

7. ***Composite***

• Razón de uso:* Se implementó para tratar de manera uniforme tanto las cuentas individuales (Cuenta) como las agrupaciones de cuentas (Portafolio). Permite a los clientes ejecutar operaciones (como getSaldo o depositar) sobre un portafolio completo como si fuera una única cuenta, simplificando el codigo cliente.

8. ***Template Method***

• Razón de uso:* Se uso para definir el esqueleto de un algoritmo en el proceso de cierre mensual (ProcesoMensualTemplate), permitiendo que las subclases (ProcesoMensualStandard , ProcesoMensualPremium) redefinen ciertos pasos (como el calculo de cargos) sin cambiar la estructura general del proceso.

FUNCIONALIDADES

Estados de cuenta: Activa, Sobregirada, Congelada, Cerrada. Cada estado controla qué operaciones están permitidas y qué acciones desencadena la cuenta.

Tipos de interés intercambiables: Estrategias para calcular intereses (Mensual, Anual, Premium). Se aplica la estrategia contratada por la cuenta.

Proceso mensual: Al fin de mes se ejecuta un proceso que:

1. Revisa sobregiros y aplica cargos.
2. Calcula e ingresa intereses según estrategia.
3. Envía un resumen (notificación).

Servicios adicionales: Servicios contratables (seguro antifraude, recompensas, alertas premium) aplicables por cuenta; combinables entre sí.

Portafolio: Un cliente puede tener múltiples cuentas y ver el portafolio consolidado (Composite).

Seguridad / acceso remoto: Requerimiento de NIP válido antes de consultar o ejecutar operaciones remotas.

Estructura de la entrega

- **README.pdf:** Este documento.
- **Diagrama de clases:** Imagen titulada Diagrama.png (o formato jpg/jpeg).
- **src:** Carpeta que contiene únicamente los archivos .java de la implementación.

Cómo Compilar y Ejecutar.

Usando Maven

bash

Compilar

mvn clean compile

Ejecutar la aplicación principal

mvn exec:java

Usando Docker

Requisitos Previos

- *Tener Docker Desktop instalado y corriendo*
- *Verificar instalación: docker --version*

Paso 1: Navegar al Directorio del Proyecto

bash

cd "c:\Users\julio\Documents\7SemestreFacCiencias\Modelado y Programación\Proyecto 1\PamparasProyecto1"

Paso 2: Construir la Imagen Docker

bash

docker build -t pumabank:1.0 .

Este proceso incluye:

- *Descarga de imagen base Java 21*
- *Descarga de dependencias Maven*

- *Compilacion del codigo fuente*
- *Empaquetado de la aplicacion*
- *Creacion de imagen optimizada (~400 MB)*

**Tiempo estimado:* 3-5 minutos (primera vez)*

Paso 3: Verificar la Imagen Creada

bash

docker images

Debería aparecer:

REPOSITORY TAG IMAGE ID CREATED SIZE

pumabank 1.0 xxxxxxxxxxxx X minutes ago ~400 MB

Paso 4: Ejecutar el Contenedor

bash

docker run -it --rm --name pumabank-app pumabank:1.0

Banderas explicadas:

- *-it : Modo interactivo con terminal*
- *--rm : Elimina el contenedor automaticamente al salir*
- *--name pumabank-app : Nombre del contenedor*

Comandos Útiles

****Ver contenedores en ejecucion:****

bash

docker ps

Ver logs:

bash

docker logs pumabank-app

Detener contenedor:

bash

docker stop pumabank-app

Eliminar imagen:

bash

docker rmi pumabank:1.0

Reconstruir después de Cambios

Si modificas el codigo fuente:

bash

docker build --no-cache -t pumabank:1.0 .

Solución de Problemas

- *Error de conexión a Docker:* Asegurate de que Docker Desktop este abierto y corriendo
- *Compilación falla:* Verifica que compile localmente primero con mvn clean compile
- *Puerto en uso:* Detener contenedor anterior con docker stop pumabank-app

Diagrama de clases

Se encuentra en Diagrama.png

Notas adicionales

- El archivo zip de entrega se llama: Proyecto1_LasPamparas.zip
- El zip **no incluye archivos .class**.
- No se usaron acentos ni la letra “ñ” en el código para evitar problemas de compilación.

Autoría y responsabilidad:
Este trabajo fue realizado únicamente por los integrantes mencionados arriba. No se incluyó a ningún alumno en más de una entrega y no se copiaron prácticas de otros equipos ni de cursos anteriores.