My Approach:

I refactored the **BankAccountController** to improve modularity, testability, fault tolerance, and overall code quality while preserving the fundamental business functionality of handling withdrawals and notifying events. I applied the below design principles:

SOLID Principles:

Split responsibilities across separate classes to adhere to the Single Responsibility and Dependency Inversion principles.

Decouple Business Logic:

Move database access and event publishing logic out of **BankAccountController**.

Dependency Injection:

Use interfaces and dependency injection for the AccountService and EventPublisher, allowing flexibility and easier testing.

Retry Logic for Robustness:

Add retry logic within the event publisher to make the system resilient to intermittent SNS service failures.

Explanation of My Implementation Choices:

Single Responsibility Principle (SRP):

Each class has a distinct responsibility. **BankAccountController** is responsible only for coordinating the withdrawal logic, **JdbcAccountService** for account balance handling, and **SnsEventPublisher** for publishing events.

Dependency Inversion Principle (DIP):

By depending on **AccountService** and **EventPublisher** interfaces rather than concrete classes, **BankAccountController** can work with any implementation of these interfaces.

Retry Logic in SnsEventPublisher:

The retry logic in **SnsEventPublisher** is isolated to make **BankAccountController** unaware of the retry mechanism, simplifying the controller and promoting encapsulation.

File Structure:

We break down the code into multiple files, each encapsulating a single class.

Explanation of SOLID Principles and Decoupling:

Single Responsibility Principle (SRP):

Each class has a single, and well-defined responsibility.

BankAccountController only coordinates the withdrawal logic.

JdbcAccountService handles data retrieval and updates.

SnsEventPublisher manages event publishing with retry logic.

Dependency Inversion Principle (DIP):

High-level classes (**BankAccountController**) depend on abstractions (**AccountService** and **EventPublisher** interfaces), not concrete implementations. This makes it easy to swap out implementations for testing or other scenarios.

Open/Closed Principle (OCP):

With interfaces and dependency injection, the code is open for extension (e.g., new event publishers) without modifying existing classes.

Interface Segregation Principle (ISP):

The **AccountService** and **EventPublisher** interfaces ensure that **BankAccountController** does not depend on unnecessary methods.

Liskov Substitution Principle (LSP):

BankAccountController works with **AccountService** and **EventPublisher** interfaces, making it interchangeable with any valid implementations of those interfaces.

Decoupling Benefits:

Database and Event Publishing Logic Removed from Controller:

Decoupling the controller from the direct use of **JdbcTemplate** and **SnsClient** allows the controller to focus on business logic.

Enhanced Testability:

Each component can be tested in isolation, and dependencies can be mocked to test each component independently.

Fault Tolerance:

With retry logic encapsulated in **SnsEventPublisher**, the code is more resilient to temporary network or service issues.

Dependency Injection and Modularity:

Dependency Injection:

Both **AccountService** and **EventPublisher** are injected into **BankAccountController**, allowing flexible substitution.

Modular Design:

The code is split into separate files for controller, service, and publisher, keeping each module clean, focused, and reusable.

File Structure:

Each class or interface resides in its own file under organized package folders:

controller:

Contains **BankAccountController** for handling withdrawal requests.

service:

Contains **AccountService** interface and **JdbcAccountService** implementation for account-related operations.

publisher:

Contains **EventPublisher** interface and **SnsEventPublisher** implementation for SNS publishing with retries

BankApplication.java:

This will contain the main method to run the application and can also serve as the Spring Boot main application class if you're using Spring Boot.

BankAccountController.java:

File path: controller/BankAccountController.java

This file handles the withdrawal endpoint and business logic.

AccountService.java:

File path: service/AccountService.java

Defines the AccountService interface.

JdbcAccountService.java:

File path: service/JdbcAccountService.java

This file implements AccountService with JDBC functionality.

EventPublisher.java:

File path: publisher/EventPublisher.java

This is the EventPublisher interface.

SnsEventPublisher.java:

File path: publisher/SnsEventPublisher.java.

This file implements the EventPublisher interface with retry logic for SNS publishing.