Outline of approach

The existing code mixes controller logic, persistence, and infrastructure concerns. To improve maintainability and scalability, I'd apply a layered architecture (Controller, Service, Repository, Event Publisher).

Improvements

- **Separation of concerns:** Extract database and SNS logic out of the controller.
- **Transaction management:** Ensure withdrawal and event publishing consistency.
- **Error handling:** Use exceptions instead of string messages in logic.
- **Observability:** Add logging and meaningful responses.
- **Data safety:** Use optimistic locking for concurrent withdrawals.
- **Flexibility:** Make SNS publishing asynchronous and abstract behind an interface.
- **Testability:** Dependency injection via constructor (instead of field injection).
- **Readability & auditability:** Use proper DTOs and JSON serialization libraries (e.g. Jackson).

Elaboration on any implementation choices

Spring @Service & @Repository: Separates business logic and DB access.

Spring Transactional: Ensures balance check and update runs atomically.

RowMapper or JPA: Instead of raw SQL strings in controller.

SNS Publisher abstraction: Decouple from AWS SDK; easier to mock/test.

Jackson ObjectMapper: Robust JSON conversion instead of string formatting.

Domain events: WithdrawalEvent as a POJO that gets serialized.

Meaningful REST responses: Use ResponseEntity with HTTP status codes.