BANKING SYSTEM PROJECT - DAY 3: JAVA IMPLEMENTATION

===================================================

OVERVIEW

--------

Day 3 focuses on implementing the core Java services for deposit and withdrawal operations. This phase establishes the foundation of the banking system's business logic with proper validation, error handling, and transaction management.

IMPLEMENTED COMPONENTS

----------------------

1. CORE ENTITY CLASSES

- Customer.java: Customer data model with Indian-specific fields

- Account.java: Account management with balance operations

- Transaction.java: Transaction recording and status management

- AuditLog.java: Comprehensive audit trail implementation

2. SERVICE LAYER IMPLEMENTATION

- DepositService.java: Deposit transaction processing

- WithdrawService.java: Withdrawal transaction processing

- TransferService.java: Inter-account transfer operations

- AuditService.java: Audit logging and compliance

3. REPOSITORY LAYER

- CustomerRepository.java: Customer data access

- AccountRepository.java: Account data access

- TransactionRepository.java: Transaction data access

- AuditLogRepository.java: Audit log data access

4. UTILITY CLASSES

- UndoRedoStack.java: Transaction reversal mechanism

- SettlementQueue.java: Batch processing queue

- ValidationUtils.java: Input validation utilities

DEPOSIT SERVICE IMPLEMENTATION

------------------------------

CLASS STRUCTURE:

public class DepositService {

private final AccountRepository accountRepository;

private final TransactionRepository transactionRepository;

private final AuditService auditService;

private final UndoRedoStack undoRedoStack;

// Constructor with dependency injection

public DepositService(AccountRepository accountRepository,

TransactionRepository transactionRepository,

AuditService auditService,

UndoRedoStack undoRedoStack)

}

CORE METHODS:

1. processDeposit Method:

- Validates account existence

- Validates deposit amount (positive, within limits)

- Updates account balance atomically

- Creates transaction record

- Logs audit trail

- Pushes transaction to undo stack

- Returns transaction details

2. validateDepositAmount Method:

- Checks for positive amount

- Validates minimum deposit amount (Rs. 0.01)

- Validates maximum deposit amount (Rs. 10,00,000)

- Throws IllegalArgumentException for invalid amounts

3. updateAccountBalance Method:

- Retrieves current account balance

- Adds deposit amount to balance

- Updates account with new balance

- Handles concurrent access with optimistic locking

BUSINESS LOGIC IMPLEMENTATION:

1. AMOUNT VALIDATION:

- Minimum deposit: Rs. 0.01

- Maximum deposit: Rs. 10,00,000 per transaction

- Positive amount validation

- Decimal precision handling (2 decimal places)

2. ACCOUNT VALIDATION:

- Account existence verification

- Account status validation (must be ACTIVE)

- Account type compatibility checks

- Customer ownership verification

3. TRANSACTION PROCESSING:

- Atomic balance updates

- Transaction ID generation

- Status tracking (PENDING -> COMPLETED)

- Error handling and rollback mechanisms

WITHDRAWAL SERVICE IMPLEMENTATION

---------------------------------

CLASS STRUCTURE:

public class WithdrawService {

private final AccountRepository accountRepository;

private final TransactionRepository transactionRepository;

private final AuditService auditService;

private final UndoRedoStack undoRedoStack;

// Constructor with dependency injection

public WithdrawService(AccountRepository accountRepository,

TransactionRepository transactionRepository,

AuditService auditService,

UndoRedoStack undoRedoStack)

}

CORE METHODS:

1. processWithdrawal Method:

- Validates account existence and status

- Validates withdrawal amount

- Checks sufficient balance availability

- Updates account balance atomically

- Creates transaction record

- Logs audit trail

- Pushes transaction to undo stack

- Returns transaction details

2. validateWithdrawalAmount Method:

- Checks for positive amount

- Validates minimum withdrawal amount (Rs. 0.01)

- Validates maximum withdrawal amount (Rs. 10,00,000)

- Throws IllegalArgumentException for invalid amounts

3. validateSufficientBalance Method:

- Retrieves current account balance

- Checks if balance >= withdrawal amount + minimum balance

- Throws InsufficientFundsException if insufficient balance

- Handles different account types with varying minimum balance requirements

BUSINESS LOGIC IMPLEMENTATION:

1. BALANCE VALIDATION:

- Sufficient funds verification

- Minimum balance maintenance

- Account type-specific rules

- Overdraft protection (if applicable)

2. WITHDRAWAL LIMITS:

- Daily withdrawal limits

- Transaction amount limits

- Account-specific restrictions

- Regulatory compliance checks

3. TRANSACTION PROCESSING:

- Atomic balance deduction

- Transaction recording

- Status management

- Error handling and recovery

TRANSFER SERVICE IMPLEMENTATION

-------------------------------

CLASS STRUCTURE:

public class TransferService {

private final AccountRepository accountRepository;

private final TransactionRepository transactionRepository;

private final AuditService auditService;

private final UndoRedoStack undoRedoStack;

// Constructor with dependency injection

public TransferService(AccountRepository accountRepository,

TransactionRepository transactionRepository,

AuditService auditService,

UndoRedoStack undoRedoStack)

}

CORE METHODS:

1. processTransfer Method:

- Validates source and destination accounts

- Validates transfer amount

- Checks sufficient balance in source account

- Performs atomic balance updates

- Creates transaction record

- Logs comprehensive audit trail

- Pushes transaction to undo stack

- Returns transaction details

2. validateTransferAmount Method:

- Checks for positive amount

- Validates minimum transfer amount (Rs. 0.01)

- Validates maximum transfer amount (Rs. 10,00,000)

- Throws IllegalArgumentException for invalid amounts

3. validateAccounts Method:

- Verifies source account existence and status

- Verifies destination account existence and status

- Prevents transfers to same account

- Validates account type compatibility

4. updateAccountBalances Method:

- Deducts amount from source account

- Adds amount to destination account

- Maintains transaction atomicity

- Handles concurrent access safely

BUSINESS LOGIC IMPLEMENTATION:

1. TRANSFER VALIDATION:

- Source account validation

- Destination account validation

- Amount validation

- Account type compatibility

- Regulatory compliance checks

2. BALANCE MANAGEMENT:

- Sufficient funds verification

- Minimum balance maintenance

- Atomic balance updates

- Rollback mechanisms

3. TRANSACTION RECORDING:

- Comprehensive transaction details

- Source and destination tracking

- Status management

- Reference number generation

AUDIT SERVICE IMPLEMENTATION

----------------------------

CLASS STRUCTURE:

public class AuditService {

private final AuditLogRepository auditLogRepository;

private final DynamoDBService dynamoDBService;

// Constructor with dependency injection

public AuditService(AuditLogRepository auditLogRepository,

DynamoDBService dynamoDBService)

}

CORE METHODS:

1. logSuccess Method:

- Creates audit log entry for successful operations

- Generates unique action ID

- Records timestamp and user details

- Stores in both MongoDB and DynamoDB

- Returns audit log reference

2. logFailure Method:

- Creates audit log entry for failed operations

- Records error details and stack trace

- Maintains failure audit trail

- Stores in both databases

- Provides debugging information

3. getAuditLogs Method:

- Retrieves audit logs by user ID

- Supports date range filtering

- Provides pagination support

- Returns formatted audit trail

BUSINESS LOGIC IMPLEMENTATION:

1. AUDIT LOGGING:

- Every operation logged with details

- User identification and timestamp

- Operation type and entity tracking

- Success/failure status recording

2. DUAL STORAGE:

- MongoDB for primary audit storage

- DynamoDB for event log storage

- Data consistency maintenance

- Backup and recovery support

3. COMPLIANCE FEATURES:

- Tamper-proof audit trail

- Regulatory compliance support

- Searchable audit records

- Export capabilities

UNDO/REDO STACK IMPLEMENTATION

------------------------------

CLASS STRUCTURE:

@Component

public class UndoRedoStack {

private final Map<String, Stack<Transaction>> undoStacks;

private final Map<String, Stack<Transaction>> redoStacks;

// Thread-safe implementation using ConcurrentHashMap

}

CORE METHODS:

1. pushUndo Method:

- Adds transaction to undo stack for specific account

- Clears redo stack when new transaction added

- Maintains transaction history per account

2. popUndo Method:

- Retrieves last transaction for reversal

- Moves transaction to redo stack

- Returns transaction for processing

3. popRedo Method:

- Retrieves last reversed transaction

- Moves transaction back to undo stack

- Returns transaction for re-execution

BUSINESS LOGIC IMPLEMENTATION:

1. TRANSACTION REVERSAL:

- Stack-based undo mechanism

- Account-specific transaction history

- Atomic reversal operations

- State consistency maintenance

2. REDO FUNCTIONALITY:

- Re-execution of reversed transactions

- State restoration capabilities

- Transaction integrity preservation

- User-friendly error handling

ERROR HANDLING IMPLEMENTATION

-----------------------------

1. CUSTOM EXCEPTIONS:

- InsufficientFundsException: For insufficient balance scenarios

- InvalidAccountException: For invalid account operations

- InvalidAmountException: For invalid transaction amounts

- TransactionFailedException: For transaction processing failures

2. EXCEPTION HANDLING STRATEGY:

- Graceful error handling

- User-friendly error messages

- Comprehensive logging

- Transaction rollback mechanisms

3. VALIDATION FRAMEWORK:

- Input validation at service layer

- Business rule validation

- Data integrity checks

- Security validation

TESTING IMPLEMENTATION

----------------------

1. UNIT TESTS:

- Service layer unit tests

- Mock repository implementations

- Business logic validation

- Error scenario testing

2. INTEGRATION TESTS:

- End-to-end transaction testing

- Database integration testing

- Service interaction testing

- Performance testing

3. TEST COVERAGE:

- Comprehensive test scenarios

- Edge case testing

- Error condition testing

- Performance benchmarking

PERFORMANCE OPTIMIZATION

------------------------

1. DATABASE OPTIMIZATION:

- Efficient query design

- Proper indexing strategy

- Connection pooling

- Caching implementation

2. MEMORY MANAGEMENT:

- Efficient object creation

- Garbage collection optimization

- Memory leak prevention

- Resource cleanup

3. CONCURRENCY HANDLING:

- Thread-safe implementations

- Optimistic locking

- Deadlock prevention

- Race condition handling

SECURITY IMPLEMENTATION

-----------------------

1. INPUT VALIDATION:

- Server-side validation

- SQL injection prevention

- XSS protection

- Data sanitization

2. ACCESS CONTROL:

- User authentication

- Authorization checks

- Role-based access

- Resource protection

3. DATA PROTECTION:

- Sensitive data encryption

- Audit trail maintenance

- Privacy compliance

- Security logging

This Java implementation provides a robust foundation for the banking system with proper business logic, error handling, and security measures.