# Mini Banking System: A Step-by-Step Java Project

This project covers core Java topics including OOP, exception handling, streams, multithreading, file I/O, logging, collections, unit testing, and design patterns.

## Step 1: Define Requirements and Use OOP Principles

Concepts: OOP (Encapsulation, Inheritance, Polymorphism)  
- Create classes: Account (base), SavingsAccount, CurrentAccount.  
- Use interfaces: TransactionService with methods deposit(), withdraw().  
- Use abstract class BankUser with subclasses Customer, Admin.

## Step 2: Implement Exception Handling

Concepts: Try-Catch, Custom Exceptions  
- Create custom exceptions like InsufficientFundsException, InvalidAccountException.  
- Wrap risky logic (e.g., withdrawals, transfers) in try-catch blocks.

## Step 3: Add File Handling for Data Persistence

Concepts: File I/O  
- Store account details and transaction logs using BufferedWriter.  
- Read existing accounts from files using BufferedReader.  
- Use Files.exists and Paths.get() for checking paths.

## Step 4: Use Collections and Generics

Concepts: List, Map, Set, and Generics  
- Maintain Map<Long, Account> for all users.  
- Use List<Transaction> to store account history.  
- Create generic method to search accounts by ID or name.

## Step 5: Use Java Streams for Data Processing

Concepts: map, filter, collect, groupingBy  
- Show account summaries: group transactions by type (credit/debit).  
- Filter accounts with balance > 10,000.  
- Calculate average transaction amount using mapToDouble().average().

## Step 6: Implement Multithreading for Concurrent Transactions

Concepts: Threads, Executors, Synchronization  
- Simulate multiple customers accessing the system using ExecutorService.  
- Ensure balance consistency with synchronized blocks or ReentrantLock.

## Step 7: Add Logging for Audit Trail

Concepts: Logging Frameworks  
- Use SLF4J or java.util.logging to log:  
 - User login/logout.  
 - Successful or failed transactions.  
 - File read/write errors.

## Step 8: Structure the Project with Proper Layers

Concepts: Package & Layering (MVC)  
- model → Account, Transaction  
- service → TransactionService, AccountService  
- controller → Main app logic  
- util → FileHandler, Logger utility

## Step 9: Write Unit Tests

Concepts: JUnit, Mockito  
- Test deposit, withdrawal, exception handling with JUnit.  
- Mock file operations or transaction services with Mockito.

## Step 10: Refactor Using Design Patterns

Concepts: Singleton, Factory, Strategy  
- Create LoggerUtility using Singleton Pattern.  
- Use Factory Pattern to create account types dynamically.  
- Implement Strategy Pattern for different transaction fee strategies (e.g., percentage-based or fixed).

## Final Output Features

- CLI-based menu system for users.  
- Secure, thread-safe transactions.  
- Exception-safe flows.  
- Persistent transaction and account history.  
- Testable and scalable structure.