Lesson 6.2: Core Banking and Legacy Systems – Practice Exercises

Prof. Dr. Joerg Osterrieder

Exercise 1: General Ledger Entries

For each banking event below, write the double-entry general ledger entries (debit and credit accounts).

1. Customer deposits €10,000 cash into their savings account
2. Bank issues a €200,000 mortgage loan to a customer
3. Customer makes a €50 purchase using their debit card
4. Bank receives €5,000 in interest payments on outstanding loans
5. Customer transfers €1,500 from their checking account to another bank
6. Bank writes off a €30,000 loan as uncollectable (bad debt)

For each entry, verify that:

- The accounting equation (Assets = Liabilities + Equity) still balances
- You can explain which sub-ledger is affected (deposits, loans, payments)

Exercise 2: Technical Debt Assessment

A mid-size European bank provides the following IT profile:

- Annual IT budget: €120M
- Maintenance spend: €90M (75% of budget)
- Number of legacy systems: 340
- Average system age: 22 years
- Number of point-to-point integrations: 1,200
- Average time to launch a new product: 14 months
- COBOL lines of code: 8 million
- COBOL developers on staff: 12 (average age: 58)

Questions:

1. Calculate the bank's "innovation ratio" (change-the-bank / total IT spend). How does it compare to cloud-native competitors?
2. If each point-to-point integration costs €15,000/year to maintain, what is the total annual integration maintenance cost?
3. Estimate the annual risk (in €) of losing 3 of the 12 COBOL developers to retirement. What assumptions did you make?
4. Propose a 3-year roadmap to reduce maintenance spend from 75% to 50% of the IT budget. What would you modernize first?

Exercise 3: Migration Strategy Selection

Three banks are planning core banking modernization. For each, recommend a migration strategy (big bang, strangler fig, or parallel run) and justify your choice.

Bank A: Retail bank, 5M customers, 40-year-old COBOL core, no in-house COBOL expertise remaining. Budget: €200M over 5 years.

Bank B: Investment bank, 2,000 institutional clients, real-time trading systems with sub-millisecond latency requirements. Core is 15 years old (Java-based). Zero downtime tolerance.

Bank C: Newly licensed digital bank (neobank), 100,000 customers, 3-year-old cloud-native core. Wants to switch from Vendor X to Vendor Y due to cost and feature limitations.

For each bank:

1. Identify the top 3 risks specific to their situation
2. Recommend a migration strategy and justify it
3. Estimate a realistic timeline
4. Describe the rollback plan if the migration fails mid-way

Exercise 4: Cloud Migration Decision Matrix

A bank has the following workloads. For each, determine whether it should be hosted on-premise, in a private cloud, or in a public cloud. Justify your choice.

Workload	Data Sensitivity	Latency Need	Scaling Pattern
Core general ledger	Very high	Medium	Steady
Fraud detection ML models	High	Very low	Bursty
Developer test environments	Low	Low	Highly variable
Customer analytics / BI	Medium	Low	Periodic spikes
High-frequency trading	High	Sub-ms	Steady
Mobile banking app backend	Medium	Low	Diurnal peaks

For each decision, address:

1. Data sovereignty constraints (assume EU GDPR applies)
2. Cost implications (CapEx vs. OpEx)
3. Regulatory acceptability

Exercise 5: Monolithic to Microservice Decomposition

A bank's monolithic core banking system handles all of the following in a single application:

Account management, payment processing, loan origination, interest calculation, KYC/AML compliance, card management, notifications, customer onboarding, regulatory reporting, and fee calculation.

Tasks:

1. Identify 5–7 bounded contexts (candidate microservices) by grouping related functions
2. For each microservice, define:
   - Its primary responsibility (single sentence)
   - The data it owns exclusively
   - Its key API endpoints (2–3 per service)
   - Which other services it needs to communicate with
3. Identify which service you would extract *first* from the monolith and explain why (consider risk, business value, and coupling)
4. Draw a dependency diagram showing how your services interact

Exercise 6: Event-Driven Architecture Design

Design an event-driven architecture for a real-time payment notification system.

Requirements:

- When a customer makes a purchase, they receive a push notification within 2 seconds
- The fraud detection system must analyze every transaction in real time
- The customer's balance must update across mobile app, web, and ATM simultaneously
- All transactions must be logged for regulatory reporting (7-year retention)

Tasks:

1. Define the key events in this system (name and payload)
2. Identify the event producers and consumers for each event
3. Choose between Apache Kafka and a traditional message queue (e.g., RabbitMQ). Justify your choice.
4. Explain how event sourcing would work for the transaction log (vs. traditional database updates)
5. What happens if the notification service goes down for 5 minutes? How does EDA handle this? (Hint: consumer offset management)

Exercise 7: Data Architecture Comparison

A bank is choosing between three data architecture approaches for its analytics platform. Compare them using the criteria below.

Criterion	Data Warehouse	Data Lake	Data Mesh
Schema approach	?	?	?
Data ownership	?	?	?
Best suited for	?	?	?
Governance model	?	?	?
Technology example	?	?	?
Risk of failure	?	?	?

Questions:

1. Fill in the table with appropriate descriptions
2. The bank has 5,000 employees across 4 business lines (retail, commercial, investment, wealth). Which approach(es) do you recommend and why?
3. How would you handle regulatory reporting data (which must be highly structured and auditable) in a data mesh model?

Exercise 8: Vendor Evaluation Scorecard

Your bank is evaluating two core banking vendors. Score each on a 1–5 scale across the criteria below and calculate a weighted total.

Criterion	Weight	Vendor A (Legacy)	Vendor B (Cloud-native)
Regulatory track record	25%	?	?
API-first architecture	20%	?	?
Total cost (5 years)	20%	?	?
Vendor financial viability	15%	?	?
Implementation speed	10%	?	?
Innovation roadmap	10%	?	?

Context for scoring:

- Vendor A: 30 years in market, 500+ bank clients, Java-based, slow release cycles, strong regulatory relationships
- Vendor B: 5 years in market, 20 bank clients, Kubernetes-native, rapid releases, VC-funded startup

Questions:

1. Score both vendors and calculate weighted totals
2. Which would you recommend for a Tier 1 bank? For a neobank? Explain the difference.
3. What due diligence would you perform on Vendor B's financial viability?

Answer Key (1/3)

Exercise 1: 1) Debit: Cash (Asset) €10K; Credit: Customer Savings (Liability) €10K. Sub-ledger: Deposits. 2) Debit: Mortgage Loans (Asset) €200K; Credit: Customer Checking (Liability) €200K. Sub-ledger: Loans. 3) Debit: Customer Checking (Liability) €50; Credit: Due to Merchant Bank (Liability) €50. Sub-ledger: Payments. 4) Debit: Cash/Due from Borrower (Asset reduction offset) €5K; Credit: Interest Income (Equity/Revenue) €5K. Sub-ledger: Loans. 5) Debit: Customer Checking (Liability) €1.5K; Credit: Nostro/Due to Other Bank (Liability) €1.5K. Sub-ledger: Payments. 6) Debit: Loan Loss Provision (Equity reduction) €30K; Credit: Mortgage Loans (Asset) €30K. Sub-ledger: Loans.

Exercise 2: 1) Innovation ratio = (€120M – €90M) / €120M = 25%. Cloud-native competitors typically achieve 50–70% innovation ratio. 2) 1,200 integrations × €15K = €18M/year on integration maintenance alone (15% of total IT budget). 3) Losing 3 of 12 COBOL devs = 25% capacity loss. If COBOL maintenance costs €20M/year (estimated from 8M LOC), losing 25% capacity risks €5M+ in delayed fixes, outsourcing premiums (€200–400/hr for COBOL contractors), and increased operational incidents. Key assumption: remaining 9 devs cannot absorb the workload. 4) Year 1: Consolidate 340 systems to ∼200 (retire redundant); reduce integrations from 1,200 to 800 (API gateway). Year 2: Migrate top-10 transaction-volume systems to cloud-native. Year 3: Automate testing and deployment. Target: 50% maintenance = €60M (saving €30M/year for innovation).

Answer Key (2/3)

Exercise 3: Bank A: Strangler fig. Risks: (1) no COBOL expertise for parallel maintenance, (2) data migration for 5M accounts, (3) regulatory approval timeline. Timeline: 4–5 years. Rollback: keep legacy running for each migrated domain until fully validated. Bank B: Parallel run. Risks: (1) sub-ms latency must be preserved, (2) zero downtime = no cutover window, (3) trading system state synchronization. Timeline: 2–3 years. Rollback: simply continue on old system (both are running). Bank C: Big bang (acceptable for small customer base). Risks: (1) data format differences between vendors, (2) API compatibility, (3) customer notification. Timeline: 6–12 months. Rollback: snapshot old system before cutover, restore if needed (100K customers manageable).

Exercise 4: Core GL: On-premise or private cloud (very high sensitivity, steady scale, regulatory scrutiny). Fraud ML: Public cloud with encryption (needs burst compute for model training, low latency for inference—consider edge inference on-premise, training in cloud). Dev/test: Public cloud (low sensitivity, highly variable, cost-optimized with spot instances). Analytics/BI: Public cloud (periodic spikes, medium sensitivity—anonymize data before cloud upload). HFT: On-premise, co-located (sub-ms latency is non-negotiable; cloud adds 1–5ms network overhead). Mobile backend: Public cloud (diurnal peaks suit auto-scaling; medium sensitivity manageable with encryption in transit and at rest).

Answer Key (3/3)

Exercise 5: Bounded contexts: (1) Account Service (CRUD, balances), (2) Payment Service (transfers, standing orders), (3) Lending Service (origination, interest calc, repayment), (4) Compliance Service (KYC/AML, screening), (5) Card Service (issuance, limits, disputes), (6) Notification Service (push, email, SMS), (7) Reporting Service (regulatory, internal BI). Extract first: Notification Service—lowest risk (non-transactional), high business value (customer experience), minimal coupling (consumes events, does not produce critical state).

Exercise 6: Events: `TransactionInitiated`, `TransactionAuthorized`, `TransactionCompleted`, `FraudAlertRaised`, `BalanceUpdated`. Kafka over RabbitMQ: Kafka provides durable, replayable event log (essential for 7-year retention and regulatory audit). RabbitMQ is fire-and-forget. Event sourcing: Store each `TransactionCompleted` event as an immutable log entry. Current balance = replay all events. Advantage: complete audit trail, temporal queries. If notification service is down: Kafka retains events. When service recovers, it resumes from its last committed offset—no events are lost. This is a key advantage of event-driven over synchronous RPC.

Exercise 7: Warehouse: Schema-on-write; centralized IT; regulatory reporting/BI; centralized governance; Snowflake/BigQuery; rigid but reliable. Lake: Schema-on-read; centralized IT; ML/exploration; light governance; S3+Spark; risk of data swamp. Mesh: Domain-defined schemas; domain teams; large orgs with many data producers; federated governance; varies; requires cultural change. Regulatory data in mesh: Create a dedicated "Regulatory Reporting" domain that consumes data products from other domains, applies strict schemas and validation, and owns the auditable output.

Exercise 8: Vendor A scores: Regulatory 5, API 2, Cost 3, Viability 5, Speed 2, Innovation 2 $\rightarrow$ Weighted = 3.35. Vendor B scores: Regulatory 2, API 5, Cost 4, Viability 2, Speed 5, Innovation 5 $\rightarrow$ Weighted = 3.45. Tier 1 bank: Vendor A (regulatory track record and viability outweigh innovation speed). Neobank: Vendor B (speed and API-first architecture are priorities; regulatory track record less critical for new entrants). Due diligence on B: Review VC funding runway, revenue growth, client retention, escrow for source code, contractual exit provisions.