Exercise 2: Bank Account Transaction Processor

AISE501 – AI in Software Engineering I

Dr. Florian Herzog

Spring Semester 2026

1 Use Case

A simple bank system maintains a set of customer accounts, each with a balance, currency, and status (`active` or `frozen`). A series of transactions is submitted for processing. The program must validate each transaction, apply valid ones, reject invalid ones, and produce output files recording the results.

2 Input Files

2.1 Account State (`accounts.json`)

A JSON file containing an array of account objects:

```
{
  "accounts": [
    {
      "account_id": "ACC-001",
      "holder": "Alice Mueller",
      "balance": 5000.00,
      "currency": "CHF",
      "status": "active"
    },
    ...
  ]
}
```

2.2 Transactions (`transactions.json`)

A JSON file containing an array of transaction objects. Each transaction has a `type` (`deposit`, `withdrawal`, or `transfer`), an `account_id`, an `amount`, and a `description`. Transfers additionally have a `to_account_id`.

3 Validation Rules

A transaction is **rejected** if any of these conditions apply:

Condition	Applies to
Account ID does not exist	All types
Account status is not `active`	All types
Amount is zero or negative	All types
Balance is less than withdrawal amount	Withdrawal, Transfer
Target account does not exist	Transfer
Target account is not `active`	Transfer
Unknown transaction type	–

4 Output

The program produces:

1. **Console output** – A summary of updated account balances, accepted transactions, and rejected transactions with reasons.
2. **Updated account state** (`accounts_updated.json`) – The accounts JSON with balances modified by accepted transactions.
3. **Transaction log** (`transaction_log.json`) – Two arrays: `accepted` and `rejected`, each transaction annotated with its `status` and (for rejections) a `reason`.

5 Expected Results

Given the provided input files, the expected outcome is:

TXN ID	Type	Result	Reason (if rejected)
TXN-001	deposit	Accepted	–
TXN-002	withdrawal	Accepted	–
TXN-003	withdrawal	Rejected	Insufficient funds
TXN-004	deposit	Rejected	Negative amount
TXN-005	deposit	Rejected	Account is frozen
TXN-006	transfer	Accepted	–
TXN-007	withdrawal	Rejected	Account not found
TXN-008	deposit	Rejected	Zero amount

6 Exercise

Two implementations are provided:

1. `bank_bad.py` – A working but poorly written version that violates many clean code and PEP 8 principles.
2. `bank_good.py` – A clean, well-structured version following PEP 8 and clean code best practices.

Tasks

1. Run both programs and verify they produce the same results.
2. Read the bad version and list all clean code / PEP 8 violations you can find.
3. For each violation, explain which principle is broken and why it makes the code harder to read or maintain.
4. Compare your list with the good version to see how each issue was resolved.

Violations to Look For

- Unused imports (`sys`, `os`, `copy`, `datetime`)
- No docstrings or module documentation
- Single-letter and abbreviated variable names (`a`, `t`, `d`, `tp`, `tid`)
- Multiple statements per line (semicolons)
- No whitespace around operators and after commas
- Manual file open/close instead of context managers (`with`)
- One giant function doing all validation (violates Single Responsibility)
- Duplicated validation logic for deposit/transfer amount checks
- No constants for file paths
- Missing `if __name__ == "__main__"` guard
- Inconsistent error handling and status assignment
- Hard-to-follow control flow with nested `if`/`elif`/`continue`