Assignment 1 — Problem 2

South African Insurance Management System

Student Name: Abishel Mathonsi
Student Number: 220196370
Qualification: Advanced Diploma in Mathematical Sciences
Module: Programming and Database Management (PDB470S)
Lecturer: Mr Rockerfeller **Institution:** Cape Peninsula University of Technology (CPUT)
Submission Date: 31 October 2025

---

1. Introduction

Ubuntu Insurance Solutions is a fictional insurer operating within South Africa's regulated environment. The company provides motor, home, life, and business insurance products. The purpose of this work is to design and implement a Python-based information system that can model policy, coverage, and claim operations in a realistic yet simplified manner.

The system demonstrates a hybrid software-engineering approach: object-oriented modelling through `dataclasses` for domain logic, combined with `pydantic.BaseModel` validation to guarantee data integrity, nested model relationships, and type safety. The design emphasises practical compliance with South African conventions such as 15 % VAT, minimum coverage limits, and the No-Claim Bonus (NCB).

---

2. Problem Analysis

Insurance administration involves a set of related entities:

- **Customer** – identifies the policyholder and captures their risk profile.
- **RiskProfile** – quantifies prior claims, age, and location rating to calculate a *risk band* (Low / Medium / High).
- **CoverageLine** – defines individual benefits such as Collision, Liability, Building, or Life Cover.
- **Policy** – groups multiple coverage lines under a term, applies fees, endorsements, and tax rules.
- **Claim** – manages the life cycle of reported losses: open → approved → settled.

A successful solution must:

1. Represent these relationships clearly;
2. Enforce validation and regulatory limits; and
3. Support computation of premiums, taxes, and claim payouts.

---

3. System Design and Architecture

The architecture is split across two notebooks:

3.1 Dataclass Layer (Domain Logic)

`dataclasses` hold state and behaviour:

- `PolicyDC`, `CoverageLineDC`, `ClaimDC`, and related classes encapsulate pricing and payout logic.
- Methods such as `annual_total_premium()` and `apply_claim()` express the mathematical business rules.
- Dunder methods (`__str__`, `__repr__`, `__eq__`) provide readable output and comparison capability.

3.2 Pydantic Layer (Validation)

`pydantic.BaseModel` classes perform pre-validation before constructing the dataclasses:

- **Field validation** – e.g., deductible ≤ limit, NCB ≤ 0.30, VAT ≤ 0.20.
- **Root validation** – enforces product-specific constraints:
  - `CAR` → must include *Liability* and *Collision*.
  - `HOME` → requires *Building* and *Contents* with minimum sums.
  - `LIFE` → requires a single *Life* cover, deductible = 0.
  - `BUSINESS` → requires *Liability* ≥ *1 000 000*.
- **Nested conversion** – validated models are converted via `.to_dc()` into executable domain objects.

This two-layer model separates *what is valid* from *how it behaves*, resulting in both correctness and flexibility.

---

4. Implementation Highlights

The system was built in Jupyter Notebook using Python 3.12.

- **Hybrid modelling:** validated Pydantic models compile into dataclasses.
- **Premium logic:** base rates × risk multiplier × (1 − NCB) + fees + VAT.
- **Claims logic:** payout = min(max(0, loss − deductible), remaining limit).
- **Endorsements:** dynamically alter coverage limits and premiums mid-term.
- **Error handling:** custom exceptions (`ValidationError`, `PolicyStateError`, `ClaimStateError`) communicate user-friendly messages during invalid actions.

Each notebook cell is annotated for clarity, mirroring professional software documentation practices.

---

5. Results and Validation

Four policy types were demonstrated:

Product	Key Rule Demonstrated	Example Claim	Outcome
CAR	Required *Liability* + *Collision*; NCB 10 %; VAT 15 %	Theft R 12 000	Payout R 9 500
HOME	Minimum Building ≥ 500 000, Contents ≥ 100 000	Contents R 15 000	Payout R 13 000
LIFE	Deductible = 0; Sum ≥ 100 000	Death R 500 000	Full Payout R 500 000

Product	Key Rule Demonstrated	Example Claim	Outcome
BUSINESS	Liability ≥ 1 000 000	Property R 150 000	Payout R 130 000

Negative test cases confirmed that invalid structures (e.g. missing Liability, under-limit Building) trigger immediate Pydantic validation errors.
Endorsement and cancellation workflows behaved as expected, updating premiums and blocking claims on inactive policies.

---

6. Reflection

The hybrid design balances expressive OOP logic with modern data-validation practices. `dataclasses` keep the computational logic transparent and lightweight, while `pydantic` prevents invalid input from ever reaching the core engine.

This approach scales well to real South African insurance environments where different product lines share 80 % of their structure but differ in rules and limits. It also demonstrates advanced Python features—type hints, enumerations, and dunder methods—fulfilling the brief's emphasis on technical depth.

---

7. Conclusion

The Ubuntu Insurance System successfully models the essentials of policy, coverage, and claim management using robust, validated Python design.
It enforces South African compliance parameters, produces realistic premium and payout calculations, and provides a modular foundation that could integrate with databases or web APIs in future work.

This submission meets the Problem 2 rubric by combining accurate problem analysis, clean implementation, validation reasoning, and professional communication of results.

---

8. Technical Appendix

Environment: Python 3.12, Jupyter Notebook
Libraries: dataclasses, pydantic, enum, typing, datetime
Notebook Files:

- `Insurance_Systems.ipynb` – Implementation & validation models
- `Insurance_Sample_Usage.ipynb` – Demonstration and results

Author: Abishel Mathonsi
Date: 31 October 2025