Markdown PDF: Export (pdf)

Problem 1: SADC Tourism Management System

Student: *Abishel Mathonsi* **Student Number:** *220196370* **Module:** PDB470S **Assessment:** Assignmnent 1 (Problem-Solving and Data-Driven Systems) **Date:** 31 October 2025

This model captures how a SADC-wide tour operator would design, price, and manage multi-country itineraries with seasonal pricing and limited capacity. I built a small set of cooperating classes: **Country** (with a month-to-season calendar and price multipliers), an abstract **Experience** with concrete types like **Safari**, **Beach**, **Cultural**, and **Adventure** (each responsible for its own pricing and per-date availability), **Package** and **PackageItem** (to assemble dated activities into an itinerary), **TouristGroup** (budget, size and preferences to validate against), and a **Booking/BookingManager** pair (to preview costs, reserve capacity and enforce rules). Together, these components price items by date and country, detect when a package is cross-border, check that a group's budget and interests are respected, and then confirm or reject a booking based on real availability—demonstrating clean encapsulation, sensible inheritance and composition, and explicit business-rule validation.

1. Introduction

Tour operators in Southern Africa manage complex combinations of destinations, seasonal demand, and inventory constraints. The goal of this work is to **model a realistic, extensible tourism domain** using **object-oriented programming (OOP)**. The system should (i) calculate date- and country-aware prices, (ii) assemble itineraries from dated activities, and (iii) validate bookings against budget, preferences, capacity, and simple cross-border rules. The final artefact is a set of Python classes (in a notebook) plus a demonstration of their behaviour.

2. Problem Analysis

Operational realities. A SADC tour operator sells experiences in multiple countries. Prices vary by **season**, certain categories (e.g., safari in winter, beach in December–January) have **category surcharges**, and each activity has **limited capacity** per date. Customers travel as **groups** with budgets and category preferences (e.g., "safari" and "adventure"). Packages often cross borders and should meet minimum trip-length expectations.

Modelling requirements.

- Represent **countries** and their seasonal calendars.
- Represent **experiences** that know how to price themselves and manage per-date availability.
- Compose **packages** as ordered sets of **dated items** with optional discounts.
- Validate bookings for **budget**, **preferences**, **minimum nights for cross-border**, and **seat availability**.
- Keep responsibilities **encapsulated** and **extensible** (easy to add a new country or experience type).

3. Solution Design

3.1 Object model (overview)

- **Country** — owns month→season and season→multiplier maps; exposes `multiplier_for(date)`.

- **Experience (abstract)** — base for all activities; encapsulates per-date availability and generic pricing.
    - **SafariExperience / BeachExperience / CulturalExperience / AdventureExperience** — override `price_for(date)` to add category-specific adjustments (e.g., safari +10% in winter; beach +8% in Dec–Jan; adventure +5% in Sep–Oct).
- **PackageItem** — a dated instance of an experience (with `nights`, `discount`).
- **Package** — collection of `PackageItem`s with `total_per_person()`, `countries`, `is_cross_border()`, and `validate(group)`.
- **TouristGroup** — carries `size`, `per_person_budget`, and `preferences`.
- **Booking / BookingManager** — preview, confirm (reserve capacity), cancel (restore capacity).

3.2 OOP principles

- **Encapsulation:** Pricing and availability live inside `Experience`; validation lives in `Package`; booking lifecycle in `BookingManager`.
- **Inheritance & Polymorphism:** Subclasses of `Experience` override `price_for(date)`.
- **Composition:** `Package` is composed of `PackageItem`s; `Booking` composes a `Package` and a `TouristGroup`.
- **Validation:** Custom exceptions (`ValidationError`, `AvailabilityError`, `BookingError`) make rules explicit.

4. Implementation

The implementation is organised in **two notebooks** to keep logic and demo cohesive yet separate:

- `Tourism_Systems.ipynb` — all **class definitions** and helper (`print_price_breakdown`).
- `Tourism_Sample_Usage.ipynb` — **demo** that loads the model with `%run -i "Tourism_Systems.ipynb"`.

Key behaviours implemented:

- Seasonal pricing via `Country.multiplier_for(date)`.
- Category adjustments inside specific `Experience` subclasses.
- Per-date capacity with `set_availability(date, seats)` and `reserve(date, qty)`.
- Package totals and cross-border detection.
- Validation: budget, preferences, and a simple "≥2 nights if cross-border" rule.
- Booking lifecycle: `preview_cost`, `confirm` (reserves seats), `cancel` (restores seats).

Why this structure works: Each rule sits with the object that "owns" it (single responsibility), and new countries or experience types are added with minimal change elsewhere (open/closed principle).

5. Demonstration Results

The following outputs were produced by `Tourism_Sample_Usage.ipynb`.

5.1 Package overview

```
Countries: ['South Africa', 'Namibia']
Cross-border?: True
Per-person total: 5645.5
```

Interpretation. The itinerary contains South Africa and Namibia, so it's cross-border. The per-person total incorporates seasonal and category rules plus discounts.

5.2 Price breakdown

```
Kruger Sunrise Safari (safari, South Africa)
  Base: 2500.00 ZAR | Season: PEAK ×1.4 | Safari winter ×1.10
  Final per-person: 3850.00 ZAR

Sossusvlei Dune Hike (adventure, Namibia)
  Base: 1800.00 NAD | Season: SHOULDER ×1.0 | Adventure Sep–Oct ×1.05 | Discount
5%
  Final per-person: 1795.50 NAD
```

Interpretation. July safari applies PEAK ×1.4 and winter ×1.10; September adventure applies ×1.05 and a 5% discount.

5.3 Booking flow

```
Preview (no seats reserved) → 33873.0
Confirm → status: CONFIRMED
Total due (per-person × group size) → 33873.0
Remaining seats (Kruger 2025-07-15) → 4
Cancel → status: CANCELLED
Restored seats (Kruger 2025-07-15) → 10
```

Interpretation. Preview validates rules without touching capacity; confirm reserves 6 seats (10→4); cancel restores per-date capacity to 10 (bounded by hard capacity).

6. Technical Reflection

Strengths.

- **Encapsulation** keeps pricing/availability logic consistent and testable.
- **Inheritance** removes duplication and centralises shared behaviour.
- **Composition** allows flexible itineraries (arbitrary dated items).
- **Validation** communicates failure modes clearly via domain exceptions.

Trade-offs.

- Currency conversion is not implemented; totals assume a single currency view.
- Cross-border policy is simple (≥2 nights); real rules could be richer (visa days, transit costs).

- Availability model is per-date and per-experience; multi-resource constraints (vehicles/guides) are not modelled.

Extensibility ideas.

- Add **FX conversion** and a "home currency" view for group totals.
- Persist bookings (SQLite/CSV) and add **reporting** (e.g., load factor per date).
- Add a lightweight **API or UI** for non-technical users.

7. Conclusion

The system successfully demonstrates how OOP principles can model a realistic SADC tourism workflow. Seasonal pricing, category surcharges, and availability are encapsulated where they belong; composition and validation ensure packages are coherent and bookable. The design is intentionally **extensible**, so new experience types, countries, or rules can be introduced with minimal change to existing code.

Appendix A: How to Run

1. Place both notebooks in the same folder: `Tourism_Systems.ipynb` and `Tourism_Sample_Usage.ipynb`.
2. Open `Tourism_Sample_Usage.ipynb` and run the first cell:

```
%run -i "Tourism_Systems.ipynb"
```

3. Run the remaining cells to reproduce the package overview, price breakdown, and booking flow.

Appendix B: Files Submitted

- `Tourism_Systems.ipynb` — implementation.
- `Tourism_Sample_Usage.ipynb` — usage demonstration.
- `Tourism_Report.md` — this report (exportable to PDF).