SADC Tourism Management System

---

1. Executive Summary

This project develops a Python-based tourism management system for SADC Tourism Connect. The system uses object-oriented programming to model countries, tourism experiences, tourist groups, and cross-border travel packages. It supports seasonal pricing, booking validation, inheritance for different experience types, and demonstrates how real-world tourism operations can be structured programmatically.

---

2. Problem Analysis

SADC Tourism Connect operates across multiple countries, each offering unique tourism products. The core challenge is creating a system that can manage the complexity of different tourism experiences, seasonal pricing differences, customer groups with varying budgets, and packages that involve multiple countries. Additionally, cross-border tourism requires extra checks, such as validating the number of countries involved and ensuring availability across experiences.

The system must allow tourists to book packages containing several activities, while ensuring business rules such as capacity, seasonal availability, and sufficient budget. Because experiences differ in nature (e.g., safari vs. beach holiday), inheritance is needed to avoid rewriting the same code multiple times. A clean design helps maintain flexibility and allows new experience types to be added in the future.

Finally, encapsulation and method design are needed to manage bookings, pricing, availability checks, and seasonal adjustments. The system should prevent over-booking and provide meaningful feedback when business rules fail.

---

3. Solution Design

The solution is built around a set of core classes: Country, TourismExperience, Safari, BeachHoliday, CulturalTour, TouristGroup, and Package.
TourismExperience is the parent class, while experience types such as Safari or BeachHoliday inherit from it. This structure promotes code reuse.

Each country contains a list of experiences, allowing the system to easily track tourism offerings. A Package groups multiple experiences and countries into a bookable product. The package also includes methods for cross-border validation and availability checking.

The pricing model uses seasonal multipliers stored inside each experience. Capacity and booking records prevent overbooking. The TouristGroup class stores preferences, group size, and budget, and includes a method to compute total budget.

All booking operations, validation checks, and cost calculations happen through the Package class, ensuring a centralized workflow—similar to how tour companies manage real travel bundles.

---

4. Implementation Steps

1. Define base classes for Country, TouristGroup, and TourismExperience.
2. Implement inheritance for Safari, BeachHoliday, and CulturalTour.
3. Add seasonal price multipliers inside the base experience class.
4. Implement booking methods with capacity tracking and validation.
5. Create the Package class to combine experiences and countries.
6. Add cross-border validation inside the package class.
7. Add availability checking logic across all experiences in a package.
8. Add a budget validation check comparing price per person to group budgets.
9. Create demonstration code to show real execution and test functionality.

---

5. Technical Justifications

Object-oriented programming is well suited for modeling real-world tourism components.
Inheritance avoids duplicated logic between different experience types.
Composition (Package containing experiences and countries) represents tourism bundles accurately.

Seasonal pricing is represented using a dictionary, allowing easy extension for more seasons.
Custom exception classes (AvailabilityError and ValidationError) make the system easier to debug and maintain.
Encapsulation ensures that business rules (e.g., capacity limits) cannot be bypassed.

The design is flexible: new experiences or package rules can be added without changing existing code. This reflects good OOP principles.

---

6. Testing & Validation

Validation was performed using printed demonstrations:

- Test bookings within capacity
- Test bookings exceeding capacity
- Test bookings exceeding tourist budget

- Test cross-border packages with duplicate countries
- Test seasonal pricing differences
- Test booking a group of different sizes

Edge cases such as zero capacity, invalid seasons, or missing countries were also checked.

---

7. Reflection & Improvements

The system works well for core tasks, but future improvements include adding database storage, a GUI interface, support for dynamic pricing based on demand, and integration with travel APIs. Additional subclasses such as hiking tours or adventure sports could also extend the model.