Car Rental System — Design & Architecture (UML)

Architecture Overview

Layered design: CLI (presentation) → Services (business rules) → Repository (SQLite persistence) → Models (domain).

Pricing uses the Strategy Pattern for flexible fee computation.

UML Class Diagram (ASCII)

```
>
> +----------------+ +----------------+ +----------------+
> | Vehicle |<>----| EconomyCar | | SUV |
> |----------------| |----------------| |----------------|
> | brand | | (inherits) | | (inherits) |
> | model | +----------------+ +----------------+
> | year | \
> | daily_rate | \ +----------------+
> | available | \----| Truck |
> +----------------+ +----------------+
>
> +----------------+ +----------------+ +----------------+
> | Customer | 1 * | Rental | * 1 | Vehicle |
> |----------------|---------|----------------|---------|----------------|
> | id, name,... | | id, dates,... | | id, ... |
> +----------------+ +----------------+ +----------------+
>
> Pricing Strategy (Strategy Pattern)
> +-------------------+ +--------------------------+
> | PricingStrategy |<-----| (Concrete Strategies) |
> |-------------------| | BaseRateStrategy |
> | +compute(...) | | WeekendSurchargeStrategy |
> +-------------------+ | LongTermDiscountStrategy |
> +--------------------------+
```

Use Cases

Actor	Use Case	Description
Customer	Register/Login	Create account, authenticate
Customer	Browse Vehicles	List/search available vehicles
Customer	Create Booking	Pick car and dates; system computes fee
Admin	Manage Cars	Add/Update/Delete vehicles
Admin	Approve/Reject	Moderate pending rental requests

Sequence (Create Booking)

```
>
> Customer -> CLI: choose 'Book a car'
> CLI -> Services: validate dates, availability
> Services -> Repository: fetch vehicle, check conflicts
> Services -> PricingStrategy: compute total fee
> Services -> Repository: persist Rental (status='PENDING')
> Repository -> CLI: rental id
> CLI -> Customer: show confirmation
```