# Telecom Plan Optimizer – Documentation

## 1. Overview

This module implements a telecom plan recommendation and cost optimization system. The system models various telecom subscription plans, evaluates customer usage requirements against these plans, and recommends the most cost-effective option that satisfies the customer’s needs, including OTT service requirements.

The architecture uses object-oriented design principles:

* An abstract base class Plan ensures consistent interface and enforcement of cost calculation logic across all plans.
* Individual plan classes extend Plan to implement specific quotas, charges, and OTT inclusions.
* A planOptimizer utility processes customer usage and produces plan recommendations.

## 2. Imports

Python

from abc import ABC, abstractmethod  
from enum import Enum, auto

* abc: Provides ABC and abstractmethod for defining abstract base classes.
* enum: Used to define OTT platforms as enumerated constants with automatic integer values.

## 3. OTT Enumeration

Python

class OTT(Enum):  
 HOTSTAR = auto()  
 PRIME = auto()  
 NETFLIX = auto()  
 SPOTIFY = auto()

Represents supported OTT (Over-The-Top) digital services. This ensures consistent references to OTT services across the system.

## 4. Customer Usage Representation

Python

class Usage:  
 def \_\_init\_\_(self, voiceMinutes, smsCount, dataMB, ottRequirements):  
 self.voiceMins = voiceMinutes  
 self.smsCount = smsCount  
 self.dataMB = dataMB  
 self.ottRequirements = ottRequirements

Represents a customer’s expected 30-day usage profile:

* voiceMins: Number of voice call minutes.
* smsCount: Number of SMS messages.
* dataMB: Data consumption in megabytes.
* ottRequirements: Set of OTT services required by the customer.

## 5. Plan Quote Representation

Python

class PlanQuote:  
 ...

Encapsulates detailed cost information for a plan after applying customer usage.

Attributes:

* planName: String, the plan’s name.
* rentalFor30Days: Normalized rental cost for 30 days.
* dataOverageCost: Cost incurred when data consumption exceeds plan quota.
* smsOverageCost: Cost incurred when SMS usage exceeds plan quota.
* voiceOverageCost: Cost incurred when voice minutes exceed plan quota.
* total: Aggregated total cost.
* OTTsOffered: Set of OTT services bundled with the plan.

Methods:

* \_\_str\_\_: Provides formatted human-readable output.

## 6. Abstract Plan Class

Python

class Plan(ABC):  
 ...

Defines a contract for all plan implementations.

Attributes:

* name: Plan name.
* cost: Plan cost for its validity period.
* validityDays: Validity period in days.
* ottIncluded: Set of OTT services included in the plan.

Methods:

* normalizedRental(): Returns plan cost normalized to 30 days.
* calculateCost(usage: Usage) -> PlanQuote: Abstract method to compute cost breakdown for given usage.

## 7. Individual Plan Implementations

Each plan extends the Plan class and implements its own cost calculation based on quotas and overage rules.

## BasicLite

* ₹249, 28 days
* Data: 1 GB/day
* Voice: 100 mins, then ₹0.75/min
* SMS: ₹0.20/SMS
* OTT: None

## Saver30

* ₹499, 30 days
* Data: 1.5 GB/day
* Voice: 300 mins, then ₹0.75/min
* SMS: 100 free, then ₹0.20/SMS
* OTT: Hotstar

## UnlimitedTalk30

* ₹650, 30 days
* Data: 5 GB total
* Voice: Unlimited
* SMS: Unlimited
* OTT: Spotify

## DataMax20

* ₹749, 20 days
* Data: Unlimited
* Voice: 100 mins, then ₹0.75/min
* SMS: Unlimited
* OTT: Hotstar

## StudentStream56

* ₹435, 30 days
* Data: 2 GB/day
* Voice: 300 mins, then ₹0.75/min
* SMS: 200 free, then ₹0.20/SMS
* OTT: Spotify

## FamilyShare30

* ₹500, 28 days
* Data: 50 GB total
* Voice: 1000 mins, then ₹0.60/min
* SMS: 500 free, then ₹0.20/SMS
* OTT: Amazon Prime

## DataMaxPlus30

* ₹1499, 30 days
* Data: Unlimited
* Voice: 300 mins, then ₹0.75/min
* SMS: 200 free, then ₹0.20/SMS
* OTT: Amazon Prime, Hotstar

## PremiumUltra30

* ₹2999, 30 days
* Data: Unlimited
* Voice: Unlimited
* SMS: Unlimited
* OTT: Netflix, Amazon Prime, Hotstar, Spotify

## 8. Plan Optimizer

Python

class planOptimizer:  
 def \_\_init\_\_(self, plans):  
 self.plans = plans  
  
 def recommend(self, usage: Usage):  
 ...

Responsible for processing usage requirements and recommending the most cost-effective plan.

Workflow:

1. Filters out any plan that does not include all of the customer’s required OTT services.
2. For eligible plans, calculates cost using plan.calculateCost(usage).
3. Collects a list of all plan quotes.
4. Determines the plan with the minimum total cost.
5. Returns (bestPlan, allQuotes).

## 9. Test Run

Python

if \_\_name\_\_ == "\_\_main\_\_":  
 ...

Includes sample usage scenarios:

* Creates Usage objects for different requirements.
* Defines available plans.
* Runs the optimizer and prints cost breakdowns.
* Displays the most cost-effective plan that meets OTT constraints.

## 10. System Flow

1. Customer specifies usage requirements via a Usage instance.
2. Optimizer filters plans to ensure OTT requirements are met.
3. For each eligible plan:
   1. Normalize its rental to 30 days.
   2. Compute overage costs for data, voice, and SMS.
   3. Generate a PlanQuote.
4. Optimizer selects the plan with the lowest total cost.
5. Results are presented in tabular (string) format.

## 11. Key Features

* Extensible architecture: New plans can easily be defined by subclassing Plan.
* Flexible usage modeling: Supports arbitrary combinations of data, voice, SMS, and OTT requirements.
* Cost optimization: Ensures selection of the cheapest plan that fulfills customer needs.
* Readable output: Provides human-readable quotations for decision making.