Arun Srinivasan

81645

# Telecom Plan Optimizer – Documentation

## Overview

This Python program helps users select the most cost-effective telecom plan based on their monthly usage and OTT (Over-The-Top) service preferences. It models various plans, calculates overage costs, and recommends the best plan that satisfies OTT requirements.

## Modules & Dependencies

- `abc`: For defining abstract base classes.
- `typing`: For type hints (`List`, `Dict`).
- `enum`: For enumerating OTT apps.

## Domain Models

### `OTTApp (Enum)`

Defines supported OTT platforms:

- `NETFLIX`
- `PRIME`
- `HOTSTAR`
- `SPOTIFY`

### `Usage`

Represents user's monthly usage:

- `minutes`: Voice call minutes.

- `sms`: Number of SMS.
- `data_mb`: Data usage in megabytes.

### `OTTRequirement`

Captures user's OTT subscription needs:

- Boolean flags for each OTT app.
- Internally stored as a dictionary mapping `OTTApp` to `bool`.

### `PlanQuote`

Stores cost breakdown for a plan:

- `plan_name`: Name of the plan.
- `rental`: Base rental cost.
- `data_overage`, `voice_overage`, `sms_overage`: Extra charges.
- `total`: Total cost.
- `__str__`: Nicely formatted output for display.

## Abstract Base Class

### `Plan (ABC)`

Base class for all telecom plans:

- `name`, `price`, `validity`, `ott_bundle`: Plan metadata.
- `price_for(usage: Usage)`: Abstract method to compute cost.
- `_scale_to_30_days(value: float)`: Normalizes values to a 30-day period.
- `provides_required_ott(requirement: OTTRequirement)`: Checks if plan satisfies OTT needs.

## Plan Subclasses

Each subclass implements `price_for()` to calculate cost based on usage and plan features.

| Class Name | OTT Included | Key Features |
|---|---|---|
| `BasicLite` | None | Low-cost, minimal data/voice/SMS |
| `Saver30` | HOTSTAR | Moderate data, voice, SMS |
| `UnlimitedTalk30` | SPOTIFY | Unlimited voice, limited data |
| `DataMax20` | HOTSTAR | High data, short validity |
| `StudentStream56` | SPOTIFY | Balanced for students |
| `FamilyShare30` | PRIME | High data/voice/SMS for families |
| `DataMaxPlus30` | PRIME, HOTSTAR | Premium data, moderate voice/SMS |
| `PremiumUltra30` | All OTTs | All-inclusive, no overage charges |

# Plan Optimizer

## `PlanOptimizer`

- Accepts a list of `Plan` objects.
- `recommend(usage, requirement)`:
  - Filters plans that meet OTT needs.
  - Calculates cost for each.
  - Returns the cheapest plan and all valid quotes.

# Main Program Flow

1. **User Input**:
   a. Voice minutes, SMS, data in GB.
   b. OTT preferences (y/n for each).
2. **Usage & Requirement Objects**:
   a. Created from user input.
3. **Plan Evaluation**:
   a. All plans are evaluated for cost.

   b. Only those meeting OTT needs are considered.
4. **Output**:
   a. Cost breakdown for each valid plan.
   b. Recommended plan with lowest total cost.

## Notes

- Data overage cost: ₹0.70 per 10MB.
- Voice overage cost: ₹0.75/min (varies by plan).
- SMS cost: ₹0.20 per SMS (after free quota).
-

ll costs normalized to 30-day period for fair comparison.

-

Code:

```
from abc import ABC,abstractmethod
from typing import List,Dict
from enum import Enum,auto

#OTT app



class Usage:
    def __init__(self,minutes:int,sms:int,data_mb:int):
        self.minutes:int=minutes
        self.sms:int=sms
        self.data_mb:int=data_mb

class OTTRequirements:
    def __init__(self,netflix=False,prime=False,hotstar=False,spotify=False):
        self.netflix=netflix
        self.prime=prime
        self.hotstar=hotstar
        self.spotify=spotify

class PlanQuote:
```

```
    def
__init__(self,plan_name,rental,data_overage,voice_overage,sms_overage,total):
        self.plan_name=plan_name
        self.rental=rental
        self.data_overage=data_overage
        self.voice_overage=voice_overage
        self.sms_overage=sms_overage
        self.total=total
    def __str__(self):
        return(f"{self.plan_name}=> Rent: Rs{self.rental},"
               f"Data{self.data_overage}=> Voice: Rs{self.voice_overage},"
               f"SMS{self.sms_overage}=> Total: Rs{self.total}"
               )

class Plan(ABC):
    def __init__(self,name:str,price:float,validity:int,ott_bundle=None):
        self.name=name
        self.price=price
        self.validity=validity
        self.ott_bundle=ott_bundle or {}
    @abstractmethod
    def price_for(self,usage:Usage)->PlanQuote:
        pass

    def _scale_to_30_days(self,value:float):
        return value*(30/self.validity)
    def provides_required_ott(self,requirement:OTTRequirements):
        for ott,needed in requirement.__dict__.items():
            if needed and not  self.ott_bundle.get(ott,False):
                return False
        return True

class BasicLite(Plan):
    def __init__(self):
        super().__init__("BasicLite",249,28,{})
    def price_for(self,usage:Usage)->PlanQuote:
        rental=self._scale_to_30_days(self.price)
        included_data=1*1024*30
        data_overage=max(0,usage.data_mb-included_data)
        data_cost=(data_overage/10)*0.70
        included_voice=self._scale_to_30_days(100)
        voice_over=max(0,usage.minutes-included_voice)
        voice_cost=voice_over+0.75
```

```
        sms_cost=usage.sms*0.20
        total=rental+data_cost+voice_cost+sms_cost
        return
PlanQuote(self.name,round(rental,2),round(data_cost,2),round(voice_cost,2),round(
sms_cost,2),round(total,2))

class Saver30(Plan):
    def __init__(self):
        super().__init__("Saver 30",499,30,{"hotstar":True})
    def price_for(self,usage:Usage)->PlanQuote:
        rental=self._scale_to_30_days(self.price)
        included_data=1.5*1024*30
        data_overage=max(0,usage.data_mb-included_data)
        data_cost=(data_overage/10)*0.70
        included_voice=self._scale_to_30_days(100)
        voice_over=max(0,usage.minutes-300)
        voice_cost=voice_over+0.75
        sms_over=max(0,usage.sms-100)

        sms_cost=sms_over*0.20
        total=rental+data_cost+voice_cost+sms_cost
        return
PlanQuote(self.name,round(rental,2),round(data_cost,2),round(voice_cost,2),round(
sms_cost,2),round(total,2))

class UnlimitedTalk30(Plan):
    def __init__(self):
        super().__init__("Unlimited Talk 30",650,30,{"spotify":True})
    def price_for(self,usage:Usage)->PlanQuote:
        rental=self._scale_to_30_days(self.price)
        included_data=5*1024*30
        data_overage=max(0,usage.data_mb-included_data)
        data_cost=(data_overage/10)*0.70
        included_voice=self._scale_to_30_days(100)
        voice_cost=0




        sms_cost=0
        total=rental+data_cost+voice_cost+sms_cost
        return
PlanQuote(self.name,round(rental,2),round(data_cost,2),round(voice_cost,2),round(
sms_cost,2),round(total,2))
```

```
class DataMax20(Plan):
    def __init__(self):
        super().__init__("Data Max 20",749,20,{"hotstar":True})
    def price_for(self,usage:Usage)->PlanQuote:
        rental=self._scale_to_30_days(self.price)
        included_data=5*1024*30
        data_overage=max(0,usage.data_mb-included_data)
        data_cost=0
        included_voice=self._scale_to_30_days(100)
        voice_over=max(0,usage.minutes-included_voice)
        voice_cost=voice_over*0.75




        sms_cost=0
        total=rental+data_cost+voice_cost+sms_cost
        return
PlanQuote(self.name,round(rental,2),round(data_cost,2),round(voice_cost,2),round(
sms_cost,2),round(total,2))

class StudentStream56(Plan):
    def __init__(self):
        super().__init__("Student Stream 56",435,30,{"spotify":True})
    def price_for(self,usage:Usage)->PlanQuote:
        rental=self._scale_to_30_days(self.price)
        included_data=2*1024*30
        data_overage=max(0,usage.data_mb-included_data)
        data_cost=(data_overage/10)*0.70
        included_voice=self._scale_to_30_days(100)
        voice_over=max(0,usage.minutes-300)
        voice_cost=voice_over*0.75
        sms_over=max(0,usage.sms-200)

        sms_cost=sms_over*0.20
        total=rental+data_cost+voice_cost+sms_cost
        return
PlanQuote(self.name,round(rental,2),round(data_cost,2),round(voice_cost,2),round(
sms_cost,2),round(total,2))

class FamilyShare30(Plan):
    def __init__(self):
        super().__init__("Family Share 30",500,28,{"prime":True})
```

```
    def price_for(self,usage:Usage)->PlanQuote:
        rental=self._scale_to_30_days(self.price)
        included_data=self._scale_to_30_days(50*1024)
        data_overage=max(0,usage.data_mb-included_data)
        data_cost=(data_overage/10)*0.70
        included_voice=self._scale_to_30_days(1000)
        voice_over=max(0,usage.minutes-included_voice)
        voice_cost=voice_over*0.60
        sms_over=max(0,usage.sms-500)

        sms_cost=sms_over*0.20
        total=rental+data_cost+voice_cost+sms_cost
        return
PlanQuote(self.name,round(rental,2),round(data_cost,2),round(voice_cost,2),round(
sms_cost,2),round(total,2))
class DataMaxPlus30(Plan):
    def __init__(self):
        super().__init__("Data Max Plus
30",1499,30,{"prime":True,"hotstar":True})
    def price_for(self,usage:Usage)->PlanQuote:
        rental=self.price

        data_cost=0

        voice_over=max(0,usage.minutes-300)
        voice_cost=voice_over*0.75
        sms_over=max(0,usage.sms-200)

        sms_cost=sms_over*0.20
        total=rental+data_cost+voice_cost+sms_cost
        return
PlanQuote(self.name,round(rental,2),round(data_cost,2),round(voice_cost,2),round(
sms_cost,2),round(total,2))
class PremiumUltra30(Plan):
    def __init__(self):
        super().__init__("Premium Ultra
30",2999,30,{"prime":True,"hotstar":True,"netflix":True,"spotify":True})
    def price_for(self,usage:Usage)->PlanQuote:
        rental=self.price

        data_cost=0

        voice_cost=0
```

```
        sms_cost=0
        total=rental+data_cost+voice_cost+sms_cost
        return PlanQuote(self.name,rental,data_cost,voice_cost,sms_cost,total)
class PlanOptimiser:
    def __init__(self,plans):
        self.plans=plans
    def recommend(self,usage:Usage,requiremnt:OTTRequirements):
        valid_quotes=[]
        for plan in self.plans:
            if plan.provides_required_ott(requiremnt):
                quote=plan.price_for(usage)
                valid_quotes.append(quote)
            if not valid_quotes:
                return None,[]
        best=min(valid_quotes,key=lambda q:q.total)
        return best,valid_quotes

if __name__=="__main__":

plans=[BasicLite(),Saver30(),UnlimitedTalk30(),DataMax20(),StudentStream56(),Fami
lyShare30(),DataMaxPlus30(),PremiumUltra30()]
    optimizer=PlanOptimiser(plans)

    usage=Usage(minutes=650,sms=300,data_mb=8*1024)
    reqs=OTTRequirements(netflix=True,prime=False,hotstar=False,spotify=False)
    best,all_quotes=optimizer.recommend(usage,reqs)
    print("Plan cost Breakdown")
    for q in all_quotes:
        print(q)
    print("Recommended Plan")
    if best:
        print(best.plan_name,"with cost Rs ",best.total)
    else:
        print("No plan meets your OTT Requirements")
```

```
PS C:\Users\arun.sr\Desktop\python\Python_Assessment> python telecom.py
Plan cost Breakdown
Recommended Plan
No plan meets your OTT Requirements
```

Question 2:

Arun Srinivasan

81645

Hypothesis :

R1 – Base Fare Rule
Every tap starts with a base fare of ₹25, regardless of line or station.

R2 – Peak Period Rule
Taps during peak hours (8:00–10:00 AM and 6:00–8:00 PM) are charged full fare. No discounts apply during this window.

R3 – Transfer Window Rule
If a rider taps within 30 minutes of a previous paid tap, the fare is ₹0. This encourages quick transfers.

R4 – Night Discount Rule
Between 10:00 AM and midnight, a 20% discount applies to the base fare unless overridden by peak or transfer rules.

R5 – Post-Midnight Discount Rule
Between 12:00 AM and 4:00 AM, a 35% discount applies to the base fare unless overridden by transfer or peak rules

2. Class Design Note

To ensure modularity and extensibility, the following classes are used:

Tap
Represents a single tap event with timestamp, station, and line.

FareRule (abstract base class)
Defines the interface for all fare rules. Each rule implements its own apply() method.

BaseFareRule, TransferRule, PeakFareRule, NightDiscountRule, PostMidnightRule
Each encapsulates one fare rule. This separation allows toggling and testing rules independently.

TariffEngine

Central engine that applies active rules in sequence. Maintains tap history and computes final fare.

Extension Points

Add new rules by subclassing FareRule.

Modify rule order or priority in TariffEngine.

Enable/disable rules via configuration flags for A/B testing.

Let me know if you'd like this formatted for a report or converted into Java next!

```
from datetime import datetime,timedelta
class Tap:
    def __init__(self,timestamp,station,line):
        self.timestamp=timestamp
        self.station=station
        self.line=line
class FareRule:
    def apply(self,tap,history,current_fare):
        return current_fare
class BaseFare(FareRule):
    def apply(self,tap,history,current_fare):
        return 25
class TransferRule(FareRule):
    def apply(self,tap,history,current_fare):
        for past_tap,fare in reversed(history):
            if fare>0 and (tap.timestamp -
past_tap.timestamp)<=timedelta(minutes=30):
                return 0
        return current_fare
class PeakPeriod(FareRule):
    def apply(self,tap,history,current_fare):
        hour=tap.timestamp.hour
        if(8<=hour<10 )or (18<=hour<20):
            return 25
        return current_fare
class NightDiscount(FareRule):
    def apply(self,tap,history,current_fare):
        hour=tap.timestamp.hour
        if 10<=hour<24:
            return current_fare*0.8
```

```
        return current_fare
class PostMidnight(FareRule):
    def apply(self,tap,history,current_fare):
        hour=tap.timestamp.hour
        if 0<=hour<4:
            return current_fare*0.65
        return current_fare



class TariffEngine:
    def __init__(self,enable_rules):
        self.rules=[]
        if enable_rules.get("R1",True):
            self.rules.append(BaseFare())
        if enable_rules.get("R2",True):
            self.rules.append(PeakPeriod())
        if enable_rules.get("R3",True):
            self.rules.append(TransferRule())
        if enable_rules.get("R4",True):
            self.rules.append(NightDiscount())
        if enable_rules.get("R5",True):
            self.rules.append(PostMidnight())
        self.history=[]
    def compute_fare(self,tap):
        fare=0
        for rule in self.rules:
            fare=rule.apply(tap,self.history,fare)
            fare=round(fare,2)
            self.history.append((tap,fare))
        return fare
def parse_log():
    raw_data = [
        ("07-01 07:20", "G", "BD"),
        ("07-01 08:01", "G", "NC"),
        ("07-01 08:30", "R", "YH"),
        ("07-01 08:32", "Y", "YH"),
        ("07-01 10:01", "R", "KL"),
        ("07-01 10:28", "Y", "NC"),
        ("07-01 10:32", "Y", "JT"),
        ("07-01 14:36", "G", "NC"),
        ("07-01 22:15", "Y", "BD"),
        ("07-01 23:58", "G", "NC"),
        ("07-02 00:45", "X", "NC"),
```

```
        ("07-02 01:10", "G", "BD"),
        ("07-02 04:01", "G", "BD"),
        ("07-02 13:05", "Y", "JT"),
        ("07-02 13:15", "G", "KL"),
        ("07-02 13:36", "G", "JT"),
        ("07-02 18:02", "Y", "BD"),
        ("07-02 18:18", "Y", "NC"),
        ("07-02 20:01", "G", "KL"),
        ("07-02 20:15", "R", "YT"),
        ("07-02 22:02", "Y", "KL"),
        ("07-02 23:15", "G", "BD"),
        ("07-03 00:20", "R", "NC"),
    ]
    taps=[]
    for entry in raw_data:
        dtr,line,station=entry
        dt=datetime.strptime(f"2025-{dtr}","%Y-%m-%d %H:%M")
        taps.append(Tap(dt,station,line))
    return taps



if __name__=="__main__":
    enable_rules={
        "R1":True,
        "R2":True,
        "R3":True,
        "R4":True,
        "R5":True

    }
    engine=TariffEngine(enable_rules)
    taps=parse_log()
    print("Datetime\tLine\tStation\tFare")
    for tap in taps:
        fare=engine.compute_fare(tap)

print(f"{tap.timestamp.strftime('%m-%d %H:%M')}\t{tap.line}\t{tap.station}\t{tap.
line}\t{tap.station}\t{fare}")
```

```
PS C:\Users\arun.sr\Desktop\python\Python_Assessment> python main.py
Datetime        Line    Station Fare
07-01 07:20     G       BD      ₹25
07-01 08:01     G       NC      ₹25
07-01 08:30     R       YH      ₹25
07-01 08:32     Y       YH      ₹25
07-01 10:01     R       KL      ₹20.0
07-01 10:28     Y       NC      ₹0.0
07-01 10:32     Y       JT      ₹20.0
07-01 14:36     G       NC      ₹20.0
07-01 22:15     Y       BD      ₹20.0
07-01 23:58     G       NC      ₹20.0
07-02 00:45     X       NC      ₹16.25
07-02 01:10     G       BD      ₹0.0
07-02 04:01     G       BD      ₹25
07-02 13:05     Y       JT      ₹20.0
07-02 13:15     G       KL      ₹0.0
07-02 13:36     G       JT      ₹20.0
07-02 18:02     Y       BD      ₹20.0
07-02 18:18     Y       NC      ₹20.0
07-02 20:01     G       KL      ₹20.0
07-02 20:15     R       YT      ₹0.0
07-02 22:02     Y       KL      ₹20.0
07-02 23:15     G       BD      ₹20.0
07-03 00:20     R       NC      ₹16.25
```