Arvind Kumar Senthil Murugan

Q1) Telecom Smart Plan Recommender (OOPS)

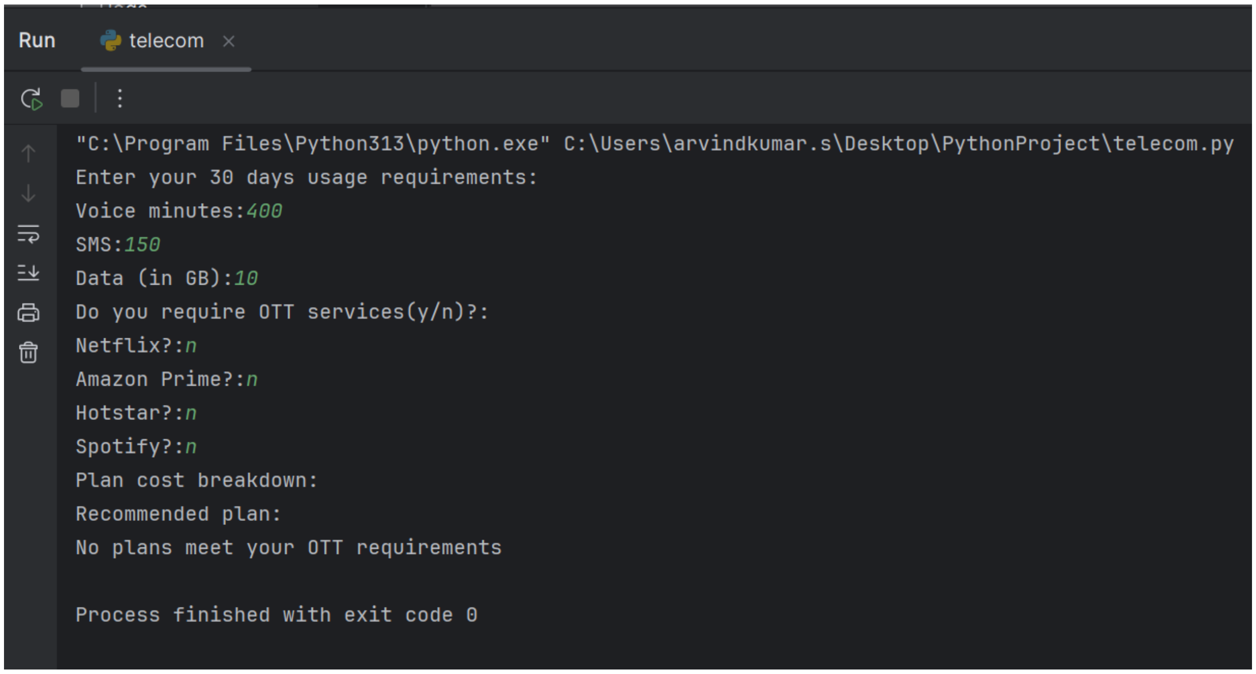
Code:

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

class OTTApp(Enum):
    NETFLIX = auto()
    PRIME = auto()
    HOTSTAR = auto()
    SPOTIFY = auto()
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 OTTRequirement:
    def __init__(self,netflix = False, prime = False, hotstar = False, spotify =
False):
        self.requirements:Dict[OTTApp,bool]={
            OTTApp.NETFLIX: netflix,
            OTTApp.PRIME: prime,
            OTTApp.HOTSTAR: hotstar,
            OTTApp.SPOTIFY: spotify,
        }
class PlanQuote:
    def
__init__(self,plan_name:str,rental:float,data_overage:float,voice_overage:float,s
ms_overage:float,total:float):
        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} => Price:Rs{self.rental}, "
               f"Data: Rs{self.data_overage}, "
               f"Voice: Rs{self.voice_overage}, "
               f"SMS: Rs{self.sms_overage}, "
               f"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) -> float:
        return value * (30 / self.validity)

    def provides_required_ott(self, requirement: OTTRequirement):
        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__("Basic Lite",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)*0.7

        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__("Saver30",499,30,[OTTApp.HOTSTAR])

    def price_for(self,usage:Usage) -> PlanQuote:
        rental = self.price
        included_data = 1.5 * 1024 * 30
        data_overage = max(0, usage.data_mb - included_data)
        data_cost = (data_overage / 10) * 0.7
        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,[OTTApp.SPOTIFY])
    def price_for(self,usage:Usage) -> PlanQuote:
        rental = self.price
        included_data = 5 * 1024
        data_overage = max(0, usage.data_mb - included_data)
        data_cost = (data_overage / 10) * 0.7
        total = rental + data_cost
        return PlanQuote(self.name, rental, round(data_cost, 2),0,0,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)
        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
        total = rental + voice_cost
        return
PlanQuote(self.name,round(rental,2),data_cost,round(voice_cost,2),0,round(total,2
))

class StudentStream56(Plan):
    def __init__(self):
        super().__init__("Student Stream 56",435,30,[OTTApp.SPOTIFY])

    def price_for(self,usage:Usage) -> PlanQuote:
        rental = self.price
        included_data = 2 * 1024 * 30
        data_overage = max(0, usage.data_mb - included_data)
        data_cost = (data_overage / 10) * 0.7
        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,rental,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,[OTTApp.PRIME])

    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) * 0.7
        included_voice = self._scale_to_30_days(1000)
        voice_over = max(0,usage.minutes - included_voice)
        voice_cost = voice_over * 0.6
        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))
```

```python
class DataMaxPlus30(Plan):
    def __init__(self):
        super().__init__("Data Max Plus
30",1499,30,[OTTApp.PRIME,OTTApp.HOTSTAR])

    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,rental,data_cost,round(voice_cost,2),round(sms_cost,2),round(
total,2))

class PremiumUltra30(Plan):
    def __init__(self):
        super().__init__("Premium Ultra
30",2999,30,[OTTApp.NETFLIX,OTTApp.PRIME,OTTApp.HOTSTAR,OTTApp.SPOTIFY])

    def price_for(self, usage: Usage) -> PlanQuote:
        rental = self.price
        return PlanQuote(self.name,rental,0,0,0,rental)

class PlanOptimizer:
    def __init__(self,plans:List[Plan]):
        self.plans = plans

    def recommend(self,usage:Usage, requirement:OTTRequirement):
        valid_quotes:List[PlanQuote] = []
        for plan in self.plans:
            if plan.provides_required_ott(requirement):
                valid_quotes.append(plan.price_for(usage))
            if not valid_quotes:
                return None, []

            best = min(valid_quotes,key = lambda q: q.total)
            return best, valid_quotes
```

```python
if __name__ == "__main__":
    plans =
[BasicLite(),Saver30(),UnlimitedTalk30(),DataMax20(),StudentStream56(),FamilyShar
e30(),DataMaxPlus30(),PremiumUltra30()]

    optimizer = PlanOptimizer(plans)

    print("Enter your 30 days usage requirements:")
    mins = int(input("Voice minutes:"))
    sms = int(input("SMS:"))
    data_gb = float(input("Data (in GB):"))
    data_mb = data_gb*1024
    print("Do you require OTT services(y/n)?:")
    netflix = input("Netflix?:").lower() =="y"
    prime = input("Amazon Prime?:").lower() =="y"
    hotstar = input("Hotstar?:").lower() =="y"
    spotify = input("Spotify?:").lower() =="y"

    usage = Usage(minutes=mins,sms= sms,data_mb = data_mb)
    reqs =
OTTRequirement(netflix=netflix,prime=prime,hotstar=hotstar,spotify=spotify)

    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 plans meet your OTT requirements")
```

Output:


```
"C:\Program Files\Python313\python.exe" C:\Users\arvindkumar.s\Desktop\PythonProject\telecom.py
Enter your 30 days usage requirements:
Voice minutes:400
SMS:150
Data (in GB):10
Do you require OTT services(y/n)?:
Netflix?:n
Amazon Prime?:n
Hotstar?:n
Spotify?:n
Plan cost breakdown:
Recommended plan:
No plans meet your OTT requirements

Process finished with exit code 0
```

Q2)City Fare OOPS:

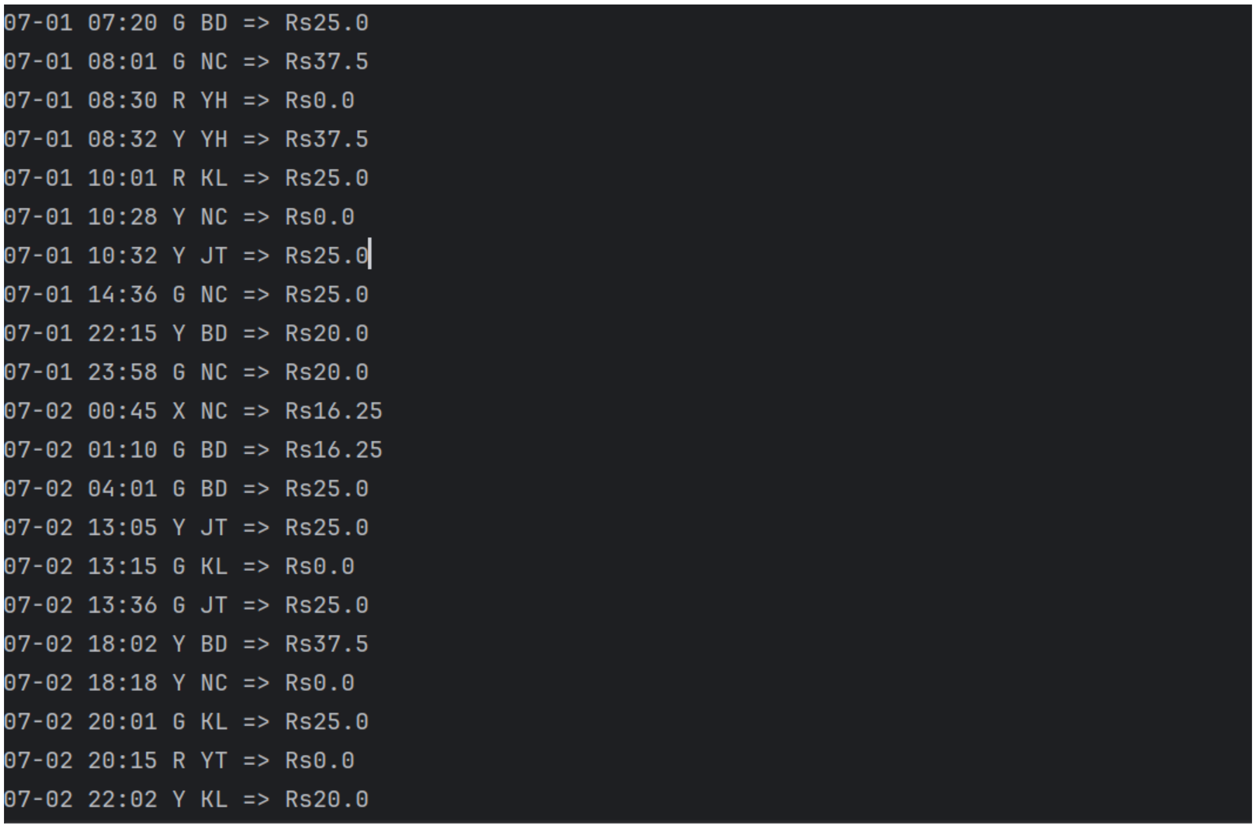
Code:

```python
from datetime import datetime,timedelta

class TapEvent:
    def __init__(self,dt_str,line,station):
        self.timestamp = datetime.strptime(dt_str, "%m-%d %H:%M")
        self.station = station
        self.line = line

class FareRule:
    def __init__(self,enabled = True):
        self.enabled = enabled

    def apply(self,event,state,current_fare):
        return current_fare
```

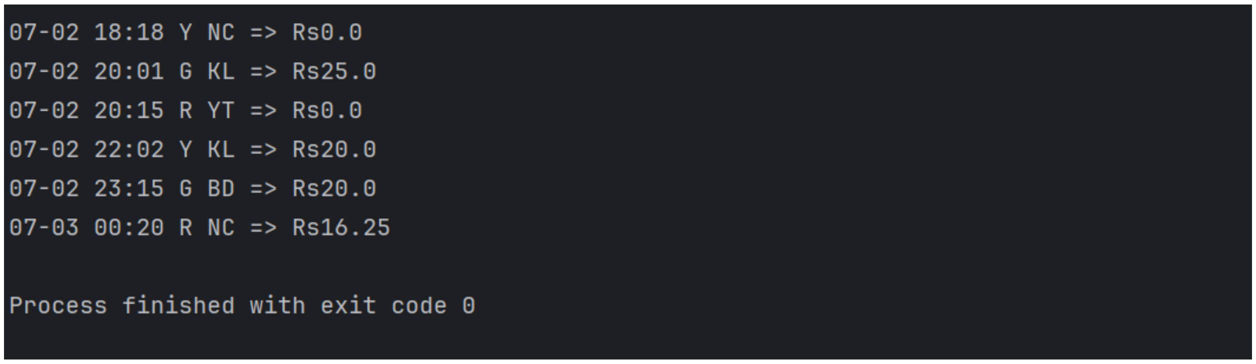
```python
class BaseFareRule(FareRule):
    def apply(self,event,state,current_fare):
        if not self.enabled:
            return current_fare
        return 25.0

class PeakSurchargeRule(FareRule):
    def apply(self,event,state,current_fare):
        if not self.enabled:
            return current_fare
        t= event.timestamp.time()
        if (t>= datetime.strptime("08:00","%H:%M").time() and t <
datetime.strptime("10:00","%H:%M").time()) \
            or (t >= datetime.strptime("18:00","%H:%M").time() and t <
datetime.strptime("20:00","%H:%M").time()):
            return 37.5
        return current_fare
class TransferRule(FareRule):
    def apply(self,event,state,current_fare):
        if not self.enabled:
            return current_fare
        last_paid_event = state.get("last_paid_event")
        last_paid_time = state.get("last_paid_time")
        if last_paid_time:
            if event.timestamp - last_paid_time <= timedelta(minutes=30):
                return 0.0
        if current_fare > 0:
            state["last_paid_event"] = event
            state["last_paid_time"] = event.timestamp
        return current_fare

class NightDiscountRule(FareRule):
    def apply(self,event,state,current_fare):
        if not self.enabled:
            return current_fare
        t = event.timestamp.time()
        if t >= datetime.strptime("22:00","%H:%M").time() or t <
datetime.strptime("00:00","%H:%M").time():
            return 20.0
        return current_fare

class PostMidnightDiscountRule(FareRule):
    def apply(self,event,state,current_fare):
        if not self.enabled:
            return current_fare
        t = event.timestamp.time()
        if t>= datetime.strptime("00:00","%H:%M").time() and t <
datetime.strptime("04:00","%H:%M").time():
            return 16.25
        return current_fare

class TariffEngine:
    def __init__(self,rules):
        self.rules = rules
        self.state = {}

    def compute_fare(self,event):
        fare = 0.0
        for rule in self.rules:
            fare = rule.apply(event,self.state,fare)
        return fare

if __name__ == "__main__":
    taps = [
        ("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"),
    ]

    engine = TariffEngine([
        BaseFareRule(True),
        PeakSurchargeRule(True),
        TransferRule(True),
        NightDiscountRule(True),
```

```python
        PostMidnightDiscountRule(True),
    ])
    for tap in taps:
        event = TapEvent(*tap)
        fare = engine.compute_fare(event)
        print(f"{event.timestamp.strftime('%m-%d %H:%M')} {event.line} {event.station} => Rs{fare}")
```

Output:



```
07-01 07:20 G BD => Rs25.0
07-01 08:01 G NC => Rs37.5
07-01 08:30 R YH => Rs0.0
07-01 08:32 Y YH => Rs37.5
07-01 10:01 R KL => Rs25.0
07-01 10:28 Y NC => Rs0.0
07-01 10:32 Y JT => Rs25.0
07-01 14:36 G NC => Rs25.0
07-01 22:15 Y BD => Rs20.0
07-01 23:58 G NC => Rs20.0
07-02 00:45 X NC => Rs16.25
07-02 01:10 G BD => Rs16.25
07-02 04:01 G BD => Rs25.0
07-02 13:05 Y JT => Rs25.0
07-02 13:15 G KL => Rs0.0
07-02 13:36 G JT => Rs25.0
07-02 18:02 Y BD => Rs37.5
07-02 18:18 Y NC => Rs0.0
07-02 20:01 G KL => Rs25.0
07-02 20:15 R YT => Rs0.0
07-02 22:02 Y KL => Rs20.0
```



```
07-02 18:18 Y NC => Rs0.0
07-02 20:01 G KL => Rs25.0
07-02 20:15 R YT => Rs0.0
07-02 22:02 Y KL => Rs20.0
07-02 23:15 G BD => Rs20.0
07-03 00:20 R NC => Rs16.25

Process finished with exit code 0
```