**Client.py**

import Animator

import Restaurant

import constants

from getpass import getpass

class Order:

dishes = []

def \_\_init\_\_(self):

self.total\_price = 0

def add\_dish(self, dish):

self.total\_price += (dish.price \* dish.quantity)

self.dishes.append(dish)

def \_\_str\_\_(self):

return "Total Price: " + str(self.total\_price) + "\nDishes: " + str(self.dishes)

def print\_header():

animator.set\_padding\_character(".")

animator.print\_full\_line()

animator.set\_padding\_character("|")

animator.print\_heading("Food booking app", spacing=40)

animator.set\_padding\_character(".")

animator.print\_full\_line()

animator.reset\_padding\_character()

def print\_menu():

choice = 0

while(choice not in [1,2,3]):

print("1. View nearby restaurants")

print("2. View order")

# print("3. Pay order")

try:

choice = int(input("Enter Selection: "))

except:

pass

return choice

def print\_restaurants():

index = 0

length = len(constants.RESTAURANT\_LIST)

choice = 0

while(choice not in range(1,length + 1)):

print()

for restaurant in constants.RESTAURANT\_LIST:

index += 1

print(str(index) + ". " + restaurant.name)

index = 0

try:

choice = int(input("Select Restaurant: "))

except:

canLoop = True

pass

return choice

def print\_restaurant\_menu(restaurant):

index = 0

length = len(restaurant.dishes)

choice = 0

canLoop = True

while(canLoop):

print()

for dish in restaurant.dishes:

index += 1

print(str(index) + ". " + dish.name.ljust(50, " ") + "- " + constants.RUPEE\_SYMBOL + str(dish.price))

index = 0

try:

choice = int(input("Select Dish: "))

if(choice not in range(1,length + 1)):

canLoop = True

print("Invalid Selection, Please select again.")

else:

dish = restaurant.dishes[choice - 1]

no\_of\_dishes = int(input("Enter no.of " + dish.name + " : "))

dish.quantity = no\_of\_dishes

order.add\_dish(dish)

canLoop = not((input("Finish Orders? Y/N: ").lower() == "y"))

except:

pass

return choice

def print\_order():

animator.print\_full\_line()

animator.print\_heading("BILL INVOICE")

animator.print\_full\_line()

index = 0

for dish in order.dishes:

index += 1

message = str(index) + ". " + dish.name.ljust(30, " ") + " - " + \

str(constants.RUPEE\_SYMBOL + str(dish.price)).rjust(10, " ") + " X " + \

str(str(dish.quantity) + "ps").ljust(10, " ") + " : " + constants.RUPEE\_SYMBOL + str(dish.quantity \* dish.price)

print(message)

print("Total Price" + str(": " + constants.RUPEE\_SYMBOL + str(order.total\_price)).rjust(56, " "))

def perform\_transaction():

select\_payment\_method()

show\_output()

def process\_payment(payment\_choice):

if payment\_choice == 1:

upi = input(constants.GET\_UPI)

elif payment\_choice == 2:

card\_number = getpass(constants.GET\_CARD\_NUMBER)

exp\_date = getpass(constants.GET\_EXPIRY\_DATE)

cvv = getpass(constants.GET\_CVV)

return

def select\_payment\_method():

payment\_choice = input(constants.PAYMENT\_METHOD)

try:

payment\_choice = int(payment\_choice)

process\_payment(payment\_choice)

except:

print(constants.INVALID\_PAYMENT\_METHOD)

select\_payment\_method()

return

def show\_output():

# Final Output and processing

animator.slow\_print(constants.UNDER\_PROCESS.format(constants.RUPEE\_SYMBOL,order.total\_price), 100)

animator.print\_processing(anim\_type=1)

animator.slow\_print(constants.TRANSACTION\_COMPLETE)

def main(order):

print\_header()

canLoop= True

while(canLoop):

choice = print\_menu()

if(choice == 1):

order = Order()

restaurant = constants.RESTAURANT\_LIST[print\_restaurants() -1]

print\_restaurant\_menu(restaurant)

print\_order()

perform\_transaction()

elif(choice == 2):

if(order and order.total\_price > 0):

print\_order()

else:

print("No Order Found")

# elif(choice==3):

# perform\_transaction()

canLoop = not(input("Exit? (Y/N)").lower() == "y")

animator.print\_full\_line()

pass

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

animator = Animator.Animation()

order = Order()

main(order)

**MenuItems.py**

class MenuItems:

name = ""

price = ""

dish\_type = ""

quantity = 1

def \_\_init\_\_(self, name, price, dish\_type=None):

self.name = name

self.price = price

self.dish\_type = dish\_type

pass

def \_\_str\_\_(self):

return "{Dish name: " + self.name + ", Price: " + str(self.price) + \

", Quantity: " + str(self.quantity) +"}"

def \_\_repr\_\_(self):

return self.\_\_str\_\_()

pass

NORTH\_INDIAN\_DISHES = [

MenuItems("Tandoori Chiken half",150),

MenuItems("Tandoori Chiken full",300),

MenuItems("Pulka",10),

MenuItems("Roti",10),

MenuItems("Butter Roti",15),

MenuItems("Naan",12),

MenuItems("Butter Naan",20),

MenuItems("Garlic Naan",18),

]

SOUTH\_INDIAN\_BREAKFAST = [

MenuItems("Oothappam",10),

MenuItems("Kal Dosa",10),

MenuItems("Plain Dosa",15),

MenuItems("Onion Dosa",20),

MenuItems("Ghee Dosa",25),

MenuItems("Chicken Dosa",35),

MenuItems("Masala Dosa",25),

MenuItems("Idly (Per piece)",8),

MenuItems("Podi Idly (Per Piece)",10),

MenuItems("Pongal",30),

]

BIRIYANI\_ITEMS = [

MenuItems("Kuska",40),

MenuItems("Chicken Biriyani",150),

MenuItems("Egg Biriyani",120),

MenuItems("Mutton Biriyani",180),

MenuItems("Beef Biriyani",200),

]

**Restaurant.py**

from MenuItems import MenuItems as item

class Restaurant:

name = ""

dishes = []

def \_\_init\_\_(self, name, dishes):

self.name = name

self.dishes = dishes

def add\_dish\_collection(self, collection):

self.dishes.extend(collection)

def add\_single\_dish(self, dish):

self.dishes.append(self)

def \_\_str\_\_(self):

return "Name: " + self.name + "\nDishes: " + str(self.dishes)

def \_\_repr\_\_(self):

return "Name: " + self.name + "\nDishes: " + str(self.dishes)

**Animator.py**

import os

import time

class Animation:

TERMINAL\_SIZE\_COLUMN = os.get\_terminal\_size().columns

PADDING\_CHAR = "\*"

ANIMATION\_PATTERN\_LIST = [

"--+++-+--+\_=\_++\_=\_+\_=",

">>>>>--------->>>>>>>>>",

"------------->>>>>",

" - - - - - - - - - - - -",

"-\_-\_-\_\_\_\_\_\_\_\_\_--\_-\_----\_\_-\_-\_\_\_-\_-\_-\_\_-\_\_\_--",

"<<<<<<<<<<<<<<<<<<<<<<<<<",

".........................",

"'-'-->.-.-->'-'-->.-.",

"############------"

".",

". . . . . .",

"..."

]

# Prints message without newline

def endless\_print(self, message):

print(message, end="")

# Prints message slowly (Slow Type Animation)

def slow\_print(self, message, speed=100, newline=True):

output = ""

for letter in message:

output += letter

print(output, flush=True, end="\r")

time.sleep(speed/1000)

if(newline):

print()

# Prints the "PADDING\_CHAR" as a complete line

def print\_full\_line(self):

print("".rjust(self.TERMINAL\_SIZE\_COLUMN, self.PADDING\_CHAR), end="")

return

# Prints a Empty Lines

def print\_clear\_line(self):

print("".rjust(self.TERMINAL\_SIZE\_COLUMN, " "), end="")

return

# Set "PADDING\_CHAR" which is used in various animation

def set\_padding\_character(self, pad\_char):

self.PADDING\_CHAR = pad\_char

# Reset "PADDING\_CHAR"

def reset\_padding\_character(self):

self.PADDING\_CHAR = "\*"

# Responsible for creating the heading bar

def print\_heading(self, heading, spacing=2):

heading\_length = len(heading) + (2\*spacing)

heading = (" " \* spacing) + heading + (" " \* spacing)

no\_of\_stars = self.TERMINAL\_SIZE\_COLUMN - heading\_length

left\_stars = 0

right\_stars = 0

if (no\_of\_stars % 2) == 0:

left\_stars = right\_stars = int(no\_of\_stars/2)

else:

left\_stars = int((no\_of\_stars - 1)/ 2)

right\_stars = left\_stars + 1

heading = "".rjust(left\_stars, self.PADDING\_CHAR) + heading + "".rjust(right\_stars, self.PADDING\_CHAR)

self.endless\_print(heading)

# Shows Processing Animation

def print\_processing(self, anim\_type=2, limit=10, anim\_data=None):

i = 0

if(anim\_type == 1):

anim\_fun = self.slash\_animation

args = 200

elif(anim\_type == 2):

anim\_fun = self.arrow\_animation

args = 30

elif(anim\_type == 3):

anim\_fun = self.custom\_animation

if anim\_data:

args = anim\_data["index"], anim\_data["max\_char"], anim\_data["speed"]

else:

args = self.ANIMATION\_PATTERN\_LIST[6], 5, 10

while i < limit:

anim\_fun(args)

i += 1

self.print\_clear\_line()

# Prints Custom animation

def custom\_animation(self, anim\_data):

message = anim\_data[0]

max\_word = anim\_data[1]

speed = anim\_data[2]

i = 0

j = i + max\_word

l = len(message)

no\_of\_times = int(l - j)

for current\_loop in range(no\_of\_times+1):

i = current\_loop

j = i + max\_word

self.slow\_print((" " \* i) + message[i:j] + (" " \* (l-j)), speed, False)

return

# Arrow =>>>>> Animation

def arrow\_animation(self, speed=30):

self.slow\_print("= ", speed, False)

self.slow\_print("=> ", speed, False)

self.slow\_print("=>> ", speed, False)

self.slow\_print(" >>> ", speed, False)

self.slow\_print(" >>>", speed, False)

self.slow\_print(" >", speed, False)

# Slash animation \-/

def slash\_animation(self, speed=200):

self.slow\_print("\\", speed, False)

self.slow\_print("-", speed, False)

self.slow\_print("/", speed, False)

**constants.py**

import MenuItems as menuitems

from Restaurant import Restaurant as restaurant

item = menuitems.MenuItems

RUPEE\_SYMBOL = str(u"\u20B9" + str(" "))

RESTAURANT\_LIST = [

restaurant("Mugal Biriyani", menuitems.BIRIYANI\_ITEMS),

restaurant("Dindugul Thalappakatti", menuitems.BIRIYANI\_ITEMS),

restaurant("Madurai Kadai", menuitems.SOUTH\_INDIAN\_BREAKFAST),

restaurant("Abrooz Meshi", menuitems.NORTH\_INDIAN\_DISHES)

]

PAYMENT\_METHOD = """

1. UPI

2. Credit/ Debit Card

Select payment method: """

INVALID\_PAYMENT\_METHOD = """Invalid Payment method, Please try again """

GET\_CARD\_NUMBER = """Enter card number: """

GET\_EXPIRY\_DATE = """Expiry Date: """

GET\_CVV = """CVV: """

GET\_UPI = """Enter UPI: """

UNDER\_PROCESS = """The payment of amount {0}{1} is under process please wait..."""

TRANSACTION\_COMPLETE = """Payment Complete..."""