Task 1

import uuid  
from datetime import date  
import json  
  
DATE\_OF\_EVENT = date(2023, 11, 15)  
TICKET\_NUMBER = 300  
BASE\_PRICE = 100  
DISCOUNT\_ADVANCED = 40  
DISCOUNT\_LATE = -10  
DISCOUNT\_STUDENT = 50  
  
class RegularTicket:  
 *"""base ticket class"""* def \_\_init\_\_(self):  
 self.\_id = uuid.uuid1()  
 self.\_price = BASE\_PRICE  
  
 @property  
 def price(self):  
 return self.\_price  
  
 def get\_ticket\_string(self):  
 return f"Ticket( {self.\_id} - {self.price} )"  
  
 def \_\_str\_\_(self):  
 return f"RegularTicket({self.\_id}) - ${self.price}"  
  
  
class AdvanceTicket(RegularTicket):  
 *"""advance ticket (purchased 60 or more days before the event)"""* def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
  
 @property  
 def price(self):  
 return self.\_price \* (BASE\_PRICE - DISCOUNT\_ADVANCED) / 100  
  
 def \_\_str\_\_(self):  
 return f"AdvanceTicket({self.\_id}) - ${self.price}"  
  
  
class LateTicket(RegularTicket):  
 *"""late ticket (purchased fewer than 10 days before the event)"""* def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
  
 @property  
 def price(self):  
 return self.\_price \* (BASE\_PRICE - DISCOUNT\_LATE) / 100  
  
 def \_\_str\_\_(self):  
 return f"LateTicket({self.\_id}) - ${self.price}"  
  
  
class StudentTicket(RegularTicket):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
  
 @property  
 def price(self):  
 return self.\_price \* (BASE\_PRICE - DISCOUNT\_STUDENT) / 100  
  
 def \_\_str\_\_(self):  
 return f"StudentTicket({self.\_id}) - ${self.price}"  
  
  
class Customer:  
 *"""class of customer"""* def \_\_init\_\_(self, name, surname, is\_student):  
  
 if not all(isinstance(i, str) for i in [name, surname]):  
 raise TypeError("Name and Surname must be String type")  
 if not isinstance(is\_student, bool):  
 raise TypeError("Is\_student must be Bool type")  
 self.\_\_name = name  
 self.\_\_surname = surname  
 self.\_\_is\_student = is\_student  
  
 @property  
 def name(self):  
 return self.\_\_name  
  
 @name.setter  
 def name(self, value):  
 if not isinstance(value, str):  
 raise TypeError("Name must be string")  
 self.\_\_name = value  
  
 @property  
 def surname(self):  
 return self.\_\_surname  
  
 @surname.setter  
 def surname(self, value):  
 if not isinstance(value, str):  
 raise TypeError("Surname must be string")  
 self.\_\_surname = value  
  
 @property  
 def is\_student(self):  
 return self.\_\_is\_student  
  
 @is\_student.setter  
 def is\_student(self, value):  
 if not isinstance(value, bool):  
 raise TypeError("Is student must be string")  
 self.\_\_is\_student = value  
  
  
class Order:  
 def \_\_init\_\_(self, customer):  
  
 if not isinstance(customer, Customer):  
 raise TypeError("customer must be Customet type")  
 self.\_\_customer = customer  
  
 @staticmethod  
 def get\_days\_to\_event():  
 today = date.today()  
 days\_to\_event = DATE\_OF\_EVENT - today  
 return days\_to\_event.days  
  
 def get\_ticket\_acording\_to\_buying\_date(self, days: int):  
 if self.\_\_customer.is\_student:  
 return StudentTicket()  
 elif days in range(11):  
 return LateTicket()  
 elif days > DISCOUNT\_ADVANCED:  
 return AdvanceTicket()  
 else:  
 return RegularTicket()  
  
 def \_\_build\_ticket\_by\_info(self, days\_to\_event, ticket):  
 *"""build ticket by student and ticket\_data"""* storage = {}  
 storage[str(ticket.\_id)] = {}  
 storage[str(ticket.\_id)]["name"] = self.\_\_customer.name  
 storage[str(ticket.\_id)]["surname"] = self.\_\_customer.surname  
 storage[str(ticket.\_id)]["is\_student"] = self.\_\_customer.is\_student  
 storage[str(ticket.\_id)]["price"] = ticket.price  
 storage[str(ticket.\_id)]["purchase\_date"] = str(date.today())  
 storage[str(ticket.\_id)]["event\_date"] = str(DATE\_OF\_EVENT)  
 storage[str(ticket.\_id)]["days\_to\_event"] = days\_to\_event  
  
 return storage  
  
 def buy\_ticket(self):  
  
 global TICKET\_NUMBER  
 days\_to\_event = Order.get\_days\_to\_event()  
 ticket = self.get\_ticket\_acording\_to\_buying\_date(days\_to\_event)  
  
 with open("ticket\_storage.json") as file:  
 storg = json.load(file)  
  
 if len(storg) >= TICKET\_NUMBER:  
 raise ValueError("Ticket ran out of stock")  
 storg.append(self.\_\_build\_ticket\_by\_info(days\_to\_event, ticket))  
  
 with open("ticket\_storage.json", "w") as file:  
 json.dump(storg, file)  
  
 def \_\_check\_ticket\_customer\_info(self, ticket, ticket\_id):  
  
 if (  
 ticket[ticket\_id]["name"] == self.\_\_customer.name  
 and ticket[ticket\_id]["surname"] == self.\_\_customer.surname  
 and ticket[ticket\_id]["is\_student"] == self.\_\_customer.is\_student  
 ):  
  
 return True  
  
 @staticmethod  
 def \_\_get\_all\_info\_about\_ticket(ticket: dict):  
 ticket\_id = list(ticket.keys())[0]  
 name = ticket[ticket\_id]["name"]  
 surname = ticket[ticket\_id]["surname"]  
 is\_student = ticket[ticket\_id]["is\_student"]  
 price = ticket[ticket\_id]["price"]  
 purchase\_date = ticket[ticket\_id]["purchase\_date"]  
 event\_date = ticket[ticket\_id]["event\_date"]  
  
 return (  
 f"\nTicket for: {name} {surname}\n"  
 f"Is\_student: {is\_student}\n"  
 f"Price: {price}\n"  
 f"Purchase\_date: {purchase\_date}\n"  
 f"Event\_date: {event\_date}\n"  
 )  
  
 def search\_ticket\_by\_ticked\_id(self, ticket\_id):  
  
 if not isinstance(ticket\_id, str):  
 raise TypeError("ticket\_id must be Sting type")  
  
 with open("ticket\_storage.json") as file:  
 ticket\_storage = json.load(file)  
 for ticket in ticket\_storage:  
 if ticket\_id in ticket:  
 if self.\_\_check\_ticket\_customer\_info(ticket, ticket\_id):  
 return Order.\_\_get\_all\_info\_about\_ticket(ticket)  
 return None  
  
  
cust1 = Customer("Andriy", "Evtyshenko", False)  
cust2 = Customer("Humpty", "Dumpty", True)  
cust3 = Customer("Hupty", "Rapty", False)  
cust4 = Customer("Kiker", "Posh", False)  
  
  
order1 = Order(customer=cust1)  
order2 = Order(customer=cust2)  
order3 = Order(customer=cust3)  
order4 = Order(customer=cust4)  
  
  
order1.buy\_ticket()  
order2.buy\_ticket()  
order3.buy\_ticket()  
order4.buy\_ticket()

JSON

ticket\_storage.json

[]

Task 2

import json  
from pprint import pprint  
from datetime import date, datetime  
import uuid  
  
PIZZA\_STORAGE = "pizza\_storage.json"  
PIZZA\_ORDERS = "pizza\_orders.json"  
ADITIONAL\_INGREDIENTS = "aditional\_ingredients.json"  
  
  
class RegularPizza:  
 *""" pizza base class """* def \_\_init\_\_(self):  
 with open(PIZZA\_STORAGE) as file:  
 self.\_pizza\_storage = json.load(file)  
  
 def get\_pizza\_price(self):  
 return self.\_pizza\_storage[self.\_\_class\_\_.\_\_name\_\_[:-5]]["price"]  
  
 def get\_ingredients(self):  
 return self.\_pizza\_storage[self.\_\_class\_\_.\_\_name\_\_[:-5]]["ingredients"]  
  
 def get\_pizza\_name(self):  
 return self.\_pizza\_storage[self.\_\_class\_\_.\_\_name\_\_[:-5]]["name"]  
  
 def \_\_str\_\_(self) -> str:  
 return f"this is {self.\_\_class\_\_.\_\_name\_\_}"  
  
  
class MondayPizza(RegularPizza):  
  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.\_pizza\_storage = self.\_pizza\_storage[0]  
  
  
class TuesdayPizza(RegularPizza):  
  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.\_pizza\_storage = self.\_pizza\_storage[1]  
  
  
class WednesdayPizza(RegularPizza):  
  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.\_pizza\_storage = self.\_pizza\_storage[2]  
  
  
class ThursdayPizza(RegularPizza):  
  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.\_pizza\_storage = self.\_pizza\_storage[3]  
  
  
class FridayPizza(RegularPizza):  
  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.\_pizza\_storage = self.\_pizza\_storage[4]  
  
  
class SaturdayPizza(RegularPizza):  
  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.\_pizza\_storage = self.\_pizza\_storage[5]  
  
  
class SundayPizza(RegularPizza):  
  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.\_pizza\_storage = self.\_pizza\_storage[6]  
  
  
class Customer:  
 *""" class of customer """* def \_\_init\_\_(self, name, surname):  
  
 if not all(isinstance(i, str) for i in [name, surname]):  
 raise TypeError("Name and Surname must be String type")  
 self.\_\_name = name  
 self.\_\_surname = surname  
  
 @property  
 def name(self):  
 return self.\_\_name  
  
 @name.setter  
 def name(self, value):  
 if not isinstance(value, str):  
 raise TypeError("Name must be string")  
 self.\_\_name = value  
  
 @property  
 def surname(self):  
 return self.\_\_surname  
  
 @surname.setter  
 def surname(self, value):  
 if not isinstance(value, str):  
 raise TypeError("Surname must be string")  
 self.\_\_surname = value  
  
  
class Order:  
 *""" class of order pizza """* def \_\_init\_\_(self, customer):  
  
 if not isinstance(customer, Customer):  
 raise TypeError("customer must be Customet type")  
 self.\_\_customer = customer  
 self.\_id = uuid.uuid1()  
  
 @staticmethod  
 def get\_day\_of\_buying(date\_buying):  
 *""" get day according to date """* return date\_buying.strftime("%A")  
  
 def add\_aditional\_ingredients(self, ingredients: dict):  
 *""" add some new products to pizza """* if not isinstance(ingredients, dict):  
 raise TypeError("ingredients\_list must be dict type")  
 with open(ADITIONAL\_INGREDIENTS) as file:  
 avaluable\_ingredients = json.load(file)  
 for element in ingredients:  
 if not avaluable\_ingredients.get(element):  
 raise ValueError("there is no such product in restourant")  
 if not (isinstance(element, str) and isinstance(avaluable\_ingredients[element], int)):  
 raise ValueError("""name of ingredient must be string type and  
 quantity must be int """)  
  
 if avaluable\_ingredients[element] - ingredients[element] < 0:  
 raise ValueError(f"There are no this amount of {element}")  
  
 avaluable\_ingredients[element] -= ingredients[element]  
  
 with open(ADITIONAL\_INGREDIENTS, "w") as file:  
 json.dump(avaluable\_ingredients, file)  
  
 return ingredients  
  
 def get\_pizza\_acording\_to\_buying\_day(self, day):  
  
 pizza\_according\_to\_day\_dict = {  
 "Monday": MondayPizza(),  
 "Tuesday": TuesdayPizza(),  
 "Wednesday": WednesdayPizza(),  
 "Thursday": ThursdayPizza(),  
 "Friday": FridayPizza(),  
 "Saturday": SaturdayPizza(),  
 "Sunday": SundayPizza(),  
  
 }  
 return pizza\_according\_to\_day\_dict[day]  
  
 def \_\_build\_pizza\_by\_info(self, day\_of\_buying, pizza, aditional\_products):  
 *""" build ticket by student and ticket\_data """* storage = {}  
 storage[str(self.\_id)] = {}  
 storage[str(self.\_id)]["name"] = self.\_\_customer.name  
 storage[str(self.\_id)]["surname"] = self.\_\_customer.surname  
 storage[str(self.\_id)]["pizza\_name"] = pizza.get\_pizza\_name()  
 storage[str(self.\_id)]["ingredients"] = pizza.get\_ingredients()  
 storage[str(self.\_id)]["aditionsl\_ingredients"] = aditional\_products  
 storage[str(self.\_id)]["price"] = pizza.get\_pizza\_price()  
 storage[str(self.\_id)]["purchase day"] = day\_of\_buying  
 return storage  
  
 def buy\_pizza(self, aditional\_products={}):  
  
 order\_date\_day = Order.get\_day\_of\_buying(date.today())  
 pizza = self.get\_pizza\_acording\_to\_buying\_day(order\_date\_day)  
  
 checked\_aditional\_products = self.add\_aditional\_ingredients(aditional\_products)  
  
 with open(PIZZA\_ORDERS) as file:  
 storg = json.load(file)  
 storg.append(self.\_\_build\_pizza\_by\_info(order\_date\_day, pizza, checked\_aditional\_products))  
  
 with open(PIZZA\_ORDERS, "w") as file:  
 json.dump(storg, file)  
  
  
customer = Customer("Andriy", "Evtyshenko")  
order = Order(customer=customer)  
order.buy\_pizza({"orange": 90})

JSON

Pizza\_storage.json

[  
 {"Monday":{  
 "name": "Hawaiian pizza",  
 "price": 175,  
 "ingredients": {  
 "sachet dried yeast": 7,  
 "lukewarm water": 250,  
 "plain flour": 400,  
 "salt": 7,  
 "olive oil": 60,  
 "bbq sauce": 250,  
 "cheese": 150,  
 "mushrooms": 150,  
 "tomato": 150,  
 "ham": 150,  
 "onion": 50  
 }  
 }  
 },  
 {"Tuesday":{  
 "name": "Chicago pizza",  
 "price": 125,  
 "ingredients": {  
 "sachet dried yeast": 7,  
 "lukewarm water": 250,  
 "plain flour": 400,  
 "salt": 7,  
 "olive oil": 60,  
 "pizza sauce": 250,  
 "cheese": 150,  
 "pineapple pieces": 150,  
 "chicken": 150  
 }  
 }  
 },  
 {"Wednesday":{  
 "name": "Ukrainian pizza",  
 "price": 95,  
 "ingredients": {  
 "sachet dried yeast": 7,  
 "lukewarm water": 250,  
 "plain flour": 400,  
 "salt": 7,  
 "olive oil": 60,  
 "bbq sauce": 250,  
 "cheese": 150,  
 "mushrooms": 150,  
 "tomato": 150,  
 "chicken": 150,  
 "onion": 50  
 }  
 }  
 },  
 {"Thursday":{  
 "name": "Combo pizza",  
 "price": 100,  
 "ingredients": {  
 "sachet dried yeast": 7,  
 "lukewarm water": 250,  
 "plain flour": 400,  
 "salt": 7,  
 "olive oil": 60,  
 "pizza sauce": 250,  
 "cheese": 150,  
 "mushrooms": 150,  
 "tomato": 150,  
 "chicken": 150,  
 "onion": 40  
 }  
 }  
 },  
 {"Friday":{  
 "name": "Roland pizza",  
 "price": 200,  
 "ingredients": {  
 "sachet dried yeast": 7,  
 "lukewarm water": 250,  
 "plain flour": 400,  
 "salt": 7,  
 "olive oil": 60,  
 "tomato paste": 250,  
 "mushrooms": 150,  
 "cheese": 150,  
 "pizza sauce": 50,  
 "chicken": 150,  
 "pepperoni": 150  
 }  
 }  
 },  
 {"Saturday":{  
 "name": "Lion pizza",  
 "price": 150,  
 "ingredients": {  
 "sachet dried yeast": 7,  
 "lukewarm water": 250,  
 "plain flour": 400,  
 "salt": 7,  
 "olive oil": 60,  
 "tomato paste": 250,  
 "cheese": 150,  
 "pepperoni": 300  
 }  
 }  
 },  
 {"Sunday":{  
 "name": "Hrono pizza",  
 "price": 175,  
 "ingredients": {  
 "sachet dried yeast": 7,  
 "lukewarm water": 250,  
 "plain flour": 400,  
 "salt": 7,  
 "olive oil": 60,  
 "bbq sauce": 250,  
 "cheese": 150,  
 "mushrooms": 150,  
 "tomato": 150,  
 "ham": 150,  
 "onion": 50  
 }  
 }  
 }  
]

Pizza\_orders.json

[]

Aditional\_ingredients.json

{"pineapple pieces": 10000, "tomato paste": 1000, "pepperoni": 1000, "sachet dried yeast": 11000, "lukewarm water": 1010, "plain flour": 11111, "salt": 800, "olive oil": 100010, "pizza sauce": 11010, "cheese": 20000, "mushrooms": 140101, "tomato": 55000, "chicken": 1555000, "onion": 4000, "meat": 7400, "apple": 2740, "banana": 5000, "orange": 2640, "strawberry": 1000, "cherry": 3000}