RELAZIONE TPSIT
“SUPERMERCATO”

Roberti Gianmarco 5AI 27/02/2024

INDICE

SPIEGAZIONE PROGETTO

Il giorno 30/01/2024 i proff. Campetella e Balducci ci hanno assegnato il progetto “Il Supermercato” da realizzare in Python. Questo programma deve simulare la gestione di un supermercato dove vengono acquistati dei beni. Il programma prevede che i prodotti del magazzino vengano immagazzinati dentro un file .json. Il supermercato ha tre casse dove i clienti devono pagare e dove i prodotti poi una volta passati in cassa vengono scalati dal magazzino. L’incasso di giornata viene salvato su un file .xml esterno. Ogni cliente verrà gestito come un Thread, dove ognuno di esso andrà in giro per il supermercato e inserirà nel proprio carrello i prodotti scelti. Una volta finita la spesa, un cliente si recherà in una delle tre casse: se libera procederà al pagamento, se occupata attenderà il completamento dei clienti avanti a lui.

Quando un cliente entra nel supermercato parte il Thread, dove con eventi random che avvengono ogni 2 secondi, prosegue il suo cammino nel supermercato.

Gli eventi da gestire sono:

1. Acquista un prodotto disponibile
2. Chiedi dove si trova un prodotto ad un commesso
3. Riponi un prodotto del tuo carrello a posto perché sbagliato

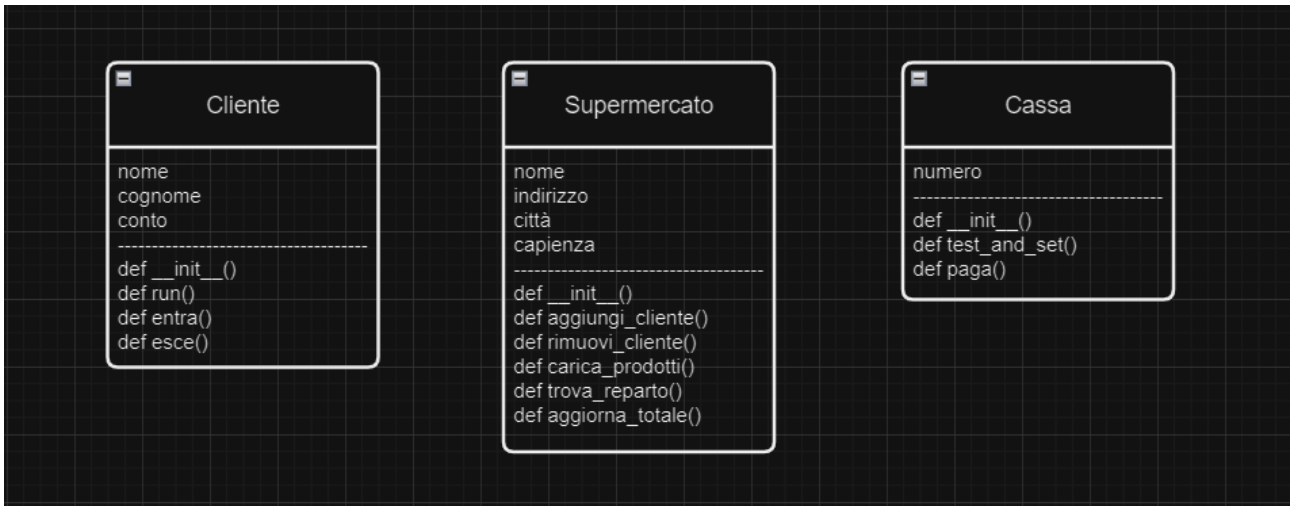
Solo dopo che sono passati 6 secondi, il cliente si dirige in cassa.

Per realizzare questo progetto ho utilizzato il software Visual Studio Code. Ho creato 3 file diversi:

- File .json, che contiene i prodotti del magazzino
- File .xml, che contiene l’incasso della giornata
- File .py, che contiene tutto il codice che serve al funzionamento del programma

Ho inizialmente creato il file magazzino.json, in cui ho inserito tutti i prodotti che idealmente sono all’interno di un supermercato. I prodotti li ho estrapolati da alcuni volantini online di supermercati della zona di Ancona e Pesaro-Urbino. Successivamente ho creato il file supermercato.py, in cui è presente la simulazione di vere e proprie spese da parte dei clienti. Infine l’incasso totale della giornata, quindi la somma di tutte le spese dei clienti, l’ho inserito all’interno del file incasso.xml.

SCHEMA UML



In questa immagine è rappresentato lo schema UML delle classi che ho utilizzato per questo progetto. La classe Cliente con i parametri nome, cognome e conto (quanti soldi ha il cliente quando entra nel supermercato) e i metodi entra, esce (per verificare l'entrata e l'uscita dei clienti dal supermercato) e run. All'interno del run è scritto tutto il codice che un cliente esegue quando parte il suo Thread. Nella classe Supermercato, invece, troviamo i parametri nome, indirizzo, città e capienza del supermercato e i metodi aggiungi cliente e rimuovi cliente, per gestire l'accesso al supermercato, carica prodotti, per caricare all'interno di una lista tutti i prodotti presenti nel magazzino, trova reparto, utile quando un cliente deve chiedere ad un commesso in che reparto si trova un determinato prodotto, e aggiorna totale, per calcolare l'incasso totale della giornata. Nella classe Cassa abbiamo invece il numero della cassa, il metodo test_and_set, utile per il controllo e l'accesso alle casse, e il metodo paga(), ovvero il metodo che viene richiamato quando un cliente ha finito la spesa e deve pagare.

CODICE

```
from threading import Thread, Semaphore
import random
import json # importazione libreria json per la gestione di file json
import threading
import time
import xml.etree.ElementTree as ET # importazione libreria xml per la gestione di
file xml

lock = threading.Lock() # creazione del lock

class Cliente(Thread):
    def __init__(self, nome, cognome, conto):
        Thread.__init__(self) # inizializzazione thread
        self.nome = nome
        self.cognome = cognome
        self.conto = conto # conto corrente cliente
        self.carrello = []
        self.pagato = False
        self.volte = 0

    def entra(self):
        if len(supermercato.clienti) < supermercato.capienza: # controllo capienza
supermercato
            supermercato.aggiungi_cliente(self) # richiamo metodo aggiungi_cliente
della classe Supermercato
            print(self.nome + " è entrato nel supermercato")
        else:
            print("Il supermercato è pieno")

    def esce(self):
        print(self.nome + " è uscito dal supermercato")
        supermercato.rimuovi_cliente(self) # cliente rimosso dal supermercato

    def run(self):
        incasso = []
        spesa = 0.0
        incasso_tot = 0.0
        quant = 0

        lock.acquire() # blocca l'accesso alla sezione di codice seguente fino a
quando il lock non viene rilasciato
        print(self.nome + " sta facendo la spesa")
        lock.release() # rilascio lock

        while(self.volte < 3): # gira 3 volte per simulare 3 eventi (6 secondi)
            scelta = random.randint(1,3) # scelta random tra 1 e 3 degli eventi
            if(scelta == 1):
                spesa = 0.0
```

```
                prodotto = random.choice(supermercato.prodotti) # scelta random di
un prodotto del supermercato
                self.carrello.append(prodotto) # aggiunta prodotto al carrello
                lock.acquire() # blocca l'accesso alla sezione di codice seguente
fino a quando il lock non viene rilasciato
                try:
                    print(f"{self.nome} sta comprando {prodotto['nome']}")
                    quant = input("Quanti ne vuoi? ") # input quantità prodotto
                finally:
                    lock.release() # rilascio il lock per permettere ad altri
thread di accedere alla sezione di codice bloccata
                prodotto['quantita'] = int(prodotto['quantita']) - int(quant) #
scalo quantità prodotto dal magazzino
                for prodotto in self.carrello:
                    spesa = float(quant) * float(prodotto['prezzo']) # calcolo
spesa di un cliente
                incasso.append(spesa) # aggiunta spesa all'incasso per quel cliente
            if(scelta == 2):
                lock.acquire()
                try:
                    richiesta = input(f"{self.nome} non sta trovando il prodotto
desiderato: ") # input prodotto desiderato ma non trovato dal cliente
                    reparto = supermercato.trova_reparto(richiesta) # richiamo
metodo trova_reparto della classe Supermercato
                    if reparto: # se il reparto è presente
                        print(f'Per trovare {richiesta} vai al reparto: {reparto}')
# stampa reparto del prodotto
                    else:
                        print('Prodotto non trovato')
                finally:
                    lock.release()
            if(scelta == 3):
                if len(self.carrello) == 0: # se il carrello è vuoto, scegli un
altro evento
                    scelta = random.randint(1,2) # scelta random tra eventi 1 e 2
                else:
                    lock.acquire()
                    try:
                        print(f"La spesa di {self.nome} è {self.carrello}")
                        err = input(f"{self.nome} ha sbagliato a prendere un
prodotto e lo rimette a posto: ") # input prodotto da rimuovere
                        prodotto_rim = None # creo prodotto da rimuovere
                        for prodotto in self.carrello: # scorro carrello
                            if(err == prodotto['nome']): # controllo se il prodotto
da rimuovere è uguale ad un prodotto nel carrello
                                prodotto_rim = prodotto # il prodotto da rimuovere
è uguale al prodotto nel carrello
                                self.carrello.remove(prodotto_rim) # rimuovo
prodotto dal carrello
```

```
                                prodotto['quantita'] = int(prodotto['quantita']) +
1
                                spesa_rimossa = float(prodotto['prezzo']) # calcolo
la spesa del prodotto rimosso
                                incasso.remove(spesa_rimossa) # sottraggo la spesa
del prodotto rimosso dall'incasso
                            else:
                                print("Prodotto non trovato")
                    finally:
                        lock.release()

            self.volte = self.volte + 1
            time.sleep(2)

        lock.acquire()
        try:
            print(self.nome + " ha finito di fare la spesa")
            incasso_tot = sum(incasso) # calcolo incasso totale della giornata di
un cliente
            t = supermercato.aggiorna_totale(incasso_tot) # richiamo metodo
aggiorna_totale della classe Supermercato per calcolare il totale di tutti i
clienti
            root = ET.Element("incasso") # creo elemento radice incasso
            totale = ET.SubElement(root, "totale") # creo elemento totale
            totale.text = str(t) # come totale scrivo l'incasso totale di tutti i
clienti
            tree = ET.ElementTree(root)
            tree.write("incasso.xml") # scrivo il file xml con l'incasso totale

            for prodotto in self.carrello:
                print(self.nome + " ha comprato " + str(prodotto['nome'])) # stampa
prodotti comprati
            print(self.nome + " ha speso " + str(incasso_tot)) # stampa spesa
totale

        finally:
            lock.release()
        cassa_r = random.choice(supermercato.casse) # scelta random di una cassa
        cassa_r.paga(self, incasso_tot) # richiamo metodo paga della classe Cassa

class Supermercato():
    def __init__(self, nome, indirizzo, città, capienza):
        self.nome = nome
        self.indirizzo = indirizzo
        self.città = città
        self.capienza = capienza
        self.clienti = [] # lista clienti del supermercato
        self.casse = [Cassa(1), Cassa(2), Cassa(3)]  # lista casse del supermercato
        self.prodotti = [] # lista prodotti disponibili nel supermercato
        self.total = 0.0
```

```
    def aggiungi_cliente(self, cliente): # metodo per aggiungere cliente alla lista
clienti
        if len(self.clienti) < self.capienza: # controllo capienza supermercato
            self.clienti.append(cliente) # aggiunta cliente alla lista clienti
        else:
            print("Il supermercato è pieno") # stampa messaggio se supermercato
pieno

    def rimuovi_cliente(self, cliente):
        self.clienti.remove(cliente) # rimozione cliente dalla lista clienti

    def carica_prodotti(self, file_path):
        with open(file_path, 'r') as f: # apri json e carica dati
            data = json.load(f)
        for prodotto in data: # scorro prodotti
            self.prodotti.append(prodotto) # aggiungo prodotto alla lista prodotti
            # print(self.prodotti)

    def trova_reparto(self, nome_prodotto):
        for prodotto in self.prodotti: # scorro prodotti
            if prodotto["nome"] == nome_prodotto: # controllo se il prodotto
inserito è presente nella lista prodotti
                return prodotto["reparto"] # prendo il reparto del prodotto
        return None

    def aggiorna_totale(self, incasso_tot):
        self.total += incasso_tot  # aggiorna il totale del supermercato
        return self.total # ritorna il totale del supermercato

class Cassa():
    def __init__(self, numero):
        self.numero = numero
        self.mutex = threading.Lock()  # creo mutex l'accesso esclusivo alla cassa
        self.libera = True  # imposta cassa come libera

    def test_and_set(self):
        with self.mutex: # acquisisce l'accesso esclusivo alla cassa
            stato = self.libera # salva il valore corrente della cassa
            self.libera = False # imposta la cassa come occupata
            return stato # ritorna il valore corrente della cassa

    def paga(self, cliente, incasso_tot):
        while True:
            if self.test_and_set():  # se la cassa è libera, cliente paga
                print(f"Cassa {self.numero} occupata") # cassa occupata perchè c'è
un cliente
                print(f"{cliente.nome} sta pagando {incasso_tot} alla cassa
{self.numero}")
                time.sleep(5)  # tempo per pagare
```

```
                if cliente.conto < incasso_tot: # controllo conto cliente
                    print(f"{cliente.nome} non ha abbastanza soldi")
                else:
                    print(f"{cliente.nome} ha pagato {incasso_tot} alla cassa
{self.numero}")
                    cliente.pagato = True # cliente ha pagato
                self.libera = True  # imposta cassa come libera
                break

print("BENVENUTI AL SUPERMERCATO!!")
nome_supermercato = input("Inserisci il nome del supermercato: ")
indirizzo = input("Inserisci l'indirizzo: ")
città = input("Inserisci la città: ")
capienza = int(input("Inserisci la capienza massima del supermercato: "))
supermercato = Supermercato(nome_supermercato, indirizzo, città, capienza)
supermercato.carica_prodotti("Supermercato/magazzino.json") # carica prodotti dal
file scritto nel percorso

while(len(supermercato.clienti) < supermercato.capienza): # gira finchè la capienza
del supermercato non è raggiunta
    nome = input("Inserisci il nome del cliente: ")
    cognome = input("Inserisci il cognome del cliente: ")
    conto = float(input("Quanto ha nel conto corrente: "))
    cliente = Cliente(nome, cognome, conto) # creazione cliente
    cliente.entra() # cliente entra nel supermercato
    print("Vuoi aggiungere un altro cliente? (s/n)") # s = si, n = no
    agg = input()
    if ((agg == 'n')): # se l'utente non vuole aggiungere altri clienti, esco dal
while
        break
    elif len(supermercato.clienti) >= supermercato.capienza: # controllo capienza
        print("Il supermercato è pieno")
        break

for cliente in supermercato.clienti: # scorro clienti
    cliente.start() # avvio thread cliente
for cliente in supermercato.clienti: # scorro clienti
    cliente.join() # attendo la terminazione del thread cliente
for cliente in supermercato.clienti: # scorro clienti
    cliente.esce() # clienti escono dal supermercato
```

SPIEGAZIONE CODICE

```
from threading import Thread, Semaphore
import random
import json # importazione libreria json per la gestione di file json
import threading
import time
import xml.etree.ElementTree as ET # importazione libreria xml per la gestione di
file xml

lock = threading.Lock() # creazione del lock
```

In questa parte di codice ho importato tutte le librerie necessarie per questo programma e ho creato anche il lock, che mi serve per gestire gli accessi a determinate parti del codice, in modo da non avere sovrapposizioni o conflitti tra clienti una volta che il programma è in esecuzione sul terminale.

```
class Cliente(Thread):
    def __init__(self, nome, cognome, conto):
        Thread.__init__(self) # inizializzazione thread
        self.nome = nome
        self.cognome = cognome
        self.conto = conto # conto corrente cliente
        self.carrello = []
        self.pagato = False
        self.volte = 0

    def entra(self):
        if len(supermercato.clienti) < supermercato.capienza: # controllo capienza
supermercato
            supermercato.aggiungi_cliente(self) # richiamo metodo aggiungi_cliente
della classe Supermercato
            print(self.nome + " è entrato nel supermercato")
        else:
            print("Il supermercato è pieno")

    def esce(self):
        print(self.nome + " è uscito dal supermercato")
        supermercato.rimuovi_cliente(self) # cliente rimosso dal supermercato
```

Successivamente troviamo la classe Cliente con il metodo per l'inizializzazione dei clienti. Ogni cliente ha un nome, cognome e il conto. La variabile conto serve per vedere se quanti soldi ha il cliente e se si può permettere quello che compra. Nella lista carrello andranno inseriti i prodotti che il cliente desidera acquistare. Il valore booleano pagato serve per controllare se il cliente ha pagato e quindi può uscire dal supermercato. La variabile volte serve per controllare che il programma, nel metodo run, giri un determinato numero di volte per poter arrivare ai 6 secondi, quando poi andrà a pagare.

Nel metodo entra ho controllato la capienza del supermercato e, se questa andava bene, aggiungevo il cliente al supermercato richiamando l'opportuno metodo. Il metodo esce, invece, serve a far uscire il cliente dal supermercato e, quindi, a rimuoverlo.

```
    def run(self):
        incasso = []
        spesa = 0.0
        incasso_tot = 0.0
        quant = 0

        lock.acquire() # blocca l'accesso alla sezione di codice seguente fino a
quando il lock non viene rilasciato
        print(self.nome + " sta facendo la spesa")
        lock.release() # rilascio lock

        while(self.volte < 3): # gira 3 volte per simulare 3 eventi (6 secondi)
            scelta = random.randint(1,3) # scelta random tra 1 e 3 degli eventi
            if(scelta == 1):
                spesa = 0.0
                prodotto = random.choice(supermercato.prodotti) # scelta random di
un prodotto del supermercato
                self.carrello.append(prodotto) # aggiunta prodotto al carrello
                lock.acquire() # blocca l'accesso alla sezione di codice seguente
fino a quando il lock non viene rilasciato
                try:
                    print(f"{self.nome} sta comprando {prodotto['nome']}")
                    quant = input("Quanti ne vuoi? ") # input quantità prodotto
                finally:
                    lock.release() # rilascio il lock per permettere ad altri
thread di accedere alla sezione di codice bloccata
                prodotto['quantita'] = int(prodotto['quantita']) - int(quant) #
scalo quantità prodotto dal magazzino
                for prodotto in self.carrello:
                    spesa = float(quant) * float(prodotto['prezzo']) # calcolo
spesa di un cliente
                incasso.append(spesa) # aggiunta spesa all'incasso per quel cliente
            if(scelta == 2):
                lock.acquire()
                try:
                    richiesta = input(f"{self.nome} non sta trovando il prodotto
desiderato: ") # input prodotto desiderato ma non trovato dal cliente
                    reparto = supermercato.trova_reparto(richiesta) # richiamo
metodo trova_reparto della classe Supermercato
                    if reparto: # se il reparto è presente
                        print(f'Per trovare {richiesta} vai al reparto: {reparto}')
# stampa reparto del prodotto
                    else:
                        print('Prodotto non trovato')
                finally:
                    lock.release()
            if(scelta == 3):
                if len(self.carrello) == 0: # se il carrello è vuoto, scegli un
altro evento
                    scelta = random.randint(1,2) # scelta random tra eventi 1 e 2
```

```
                else:
                    lock.acquire()
                    try:
                        print(f"La spesa di {self.nome} è {self.carrello}")
                        err = input(f"{self.nome} ha sbagliato a prendere un
prodotto e lo rimette a posto: ") # input prodotto da rimuovere
                        prodotto_rim = None # creo prodotto da rimuovere
                        for prodotto in self.carrello: # scorro carrello
                            if(err == prodotto['nome']): # controllo se il prodotto
da rimuovere è uguale ad un prodotto nel carrello
                                prodotto_rim = prodotto # il prodotto da rimuovere
è uguale al prodotto nel carrello
                                self.carrello.remove(prodotto_rim) # rimuovo
prodotto dal carrello
                                prodotto['quantita'] = int(prodotto['quantita']) +
1
                                spesa_rimossa = float(prodotto['prezzo']) # calcolo
la spesa del prodotto rimosso
                                incasso.remove(spesa_rimossa) # sottraggo la spesa
del prodotto rimosso dall'incasso
                            else:
                                print("Prodotto non trovato")
                    finally:
                        lock.release()

            self.volte = self.volte + 1
            time.sleep(2)

        lock.acquire()
        try:
            print(self.nome + " ha finito di fare la spesa")
            incasso_tot = sum(incasso) # calcolo incasso totale della giornata di
un cliente
            t = supermercato.aggiorna_totale(incasso_tot) # richiamo metodo
aggiorna_totale della classe Supermercato per calcolare il totale di tutti i
clienti
            root = ET.Element("incasso") # creo elemento radice incasso
            totale = ET.SubElement(root, "totale") # creo elemento totale
            totale.text = str(t) # come totale scrivo l'incasso totale di tutti i
clienti
            tree = ET.ElementTree(root)
            tree.write("incasso.xml") # scrivo il file xml con l'incasso totale

            for prodotto in self.carrello:
                print(self.nome + " ha comprato " + str(prodotto['nome'])) # stampa
prodotti comprati
            print(self.nome + " ha speso " + str(incasso_tot)) # stampa spesa
totale

        finally:
```

```
            lock.release()
        cassa_r = random.choice(supermercato.casse) # scelta random di una cassa
        cassa_r.paga(self, incasso_tot) # richiamo metodo paga della classe Cassa
```

Questo è il metodo run, ovvero il metodo che i thread eseguono una volta che sono partiti. In questo metodo ho inserito un while di modo che ogni cliente faccia almeno 3 eventi, di modo che dopo 6 secondi possa andare a pagare. All'interno di questo ciclo troviamo un menu in cui si presentano diverse opzioni che vengono scelte randomicamente:

- Opzione 1: cliente sceglie random un prodotto dal magazzino e lo aggiunge al carrello. Qui viene scalata la quantità dal magazzino e viene calcolata la spesa per quel prodotto, data dalla quantità (inserita dall'utente) e il prezzo;
- Opzione 2: cliente chiede ad un commesso dove si trova un prodotto. Qui viene mostrato a schermo il reparto dove si trova quel prodotto;
- Opzione 3: cliente mette a posto un prodotto presente nel suo carrello.

Finito il while, il cliente ha finito di fare la spesa. A questo punto calcolo sia la spesa totale di ogni cliente che l'incasso totale fatto dal supermercato. Quest'ultimo lo salvo all'interno di un file xml. Avviene poi la stampa dello scontrino con nomi e totale dei prodotti. Infine avviene il pagamento presso una cassa scelta in modo randomico.

```
class Supermercato():
    def __init__(self, nome, indirizzo, città, capienza):
        self.nome = nome
        self.indirizzo = indirizzo
        self.città = città
        self.capienza = capienza
        self.clienti = [] # lista clienti del supermercato
        self.casse = [Cassa(1), Cassa(2), Cassa(3)]  # lista casse del supermercato
        self.prodotti = [] # lista prodotti disponibili nel supermercato
        self.total = 0.0

    def aggiungi_cliente(self, cliente): # metodo per aggiungere cliente alla lista
clienti
        if len(self.clienti) < self.capienza: # controllo capienza supermercato
            self.clienti.append(cliente) # aggiunta cliente alla lista clienti
        else:
            print("Il supermercato è pieno") # stampa messaggio se supermercato
pieno

    def rimuovi_cliente(self, cliente):
        self.clienti.remove(cliente) # rimozione cliente dalla lista clienti
```

Questa è la classe Supermercato dove abbiamo l'inizializzazione di esso. Il supermercato ha come attributi il nome, l'indirizzo, la città e la capienza. Vengono poi inizializzati anche la lista dei clienti, la lista delle casse, la lista dei prodotti e il totale giornaliero. Il metodo aggiungi cliente controlla la capienza del supermercato e aggiunge il cliente dentro la lista clienti. Il metodo rimuovi, invece, rimuove un cliente dalla lista.

```
def carica_prodotti(self, file_path):
        with open(file_path, 'r') as f: # apri json e carica dati
            data = json.load(f)
        for prodotto in data: # scorro prodotti
            self.prodotti.append(prodotto) # aggiungo prodotto alla lista prodotti
            # print(self.prodotti)
```

Il metodo carica prodotti serve per caricare i prodotti da un file .json dentro ad una lista prodotti. Nel file path andrà poi scritto il percorso del file. Carico i dati dentro data e poi scorro tutti i prodotti e li aggiungo in una lista.

```
    def trova_reparto(self, nome_prodotto):
        for prodotto in self.prodotti: # scorro prodotti
            if prodotto["nome"] == nome_prodotto: # controllo se il prodotto
inserito è presente nella lista prodotti
                return prodotto["reparto"] # prendo il reparto del prodotto
        return None
```

Questo metodo serve per trovare il reparto del prodotto richiesto dal cliente al commesso. Cerco nella lista dei prodotti se quello inserito dall'utente è presente, se è cosi ritorno il reparto.

```
    def aggiorna_totale(self, incasso_tot):
        self.total += incasso_tot  # aggiorna il totale del supermercato
        return self.total # ritorna il totale del supermercato
```

Questo metodo aggiorna l'incasso totale giornaliero del supermercato. La variabile passata come parametro è l'incasso totale di un singolo cliente e quindi la aggiorno ogni volta per tutti i clienti e il totale lo salvo in total.

```
class Cassa():
    def __init__(self, numero):
        self.numero = numero
        self.mutex = threading.Lock()  # creo mutex l'accesso esclusivo alla cassa
        self.libera = True  # imposta cassa come libera

    def test_and_set(self):
        with self.mutex: # acquisisce l'accesso esclusivo alla cassa
            stato = self.libera # salva il valore corrente della cassa
            self.libera = False # imposta la cassa come occupata
            return stato # ritorna il valore corrente della cassa

    def paga(self, cliente, incasso_tot):
        while True:
            if self.test_and_set():  # se la cassa è libera, cliente paga
                print(f"Cassa {self.numero} occupata") # cassa occupata perchè c'è
un cliente
                print(f"{cliente.nome} sta pagando {incasso_tot} alla cassa
{self.numero}")
                time.sleep(5)  # tempo per pagare
```

```
                if cliente.conto < incasso_tot: # controllo conto cliente
                    print(f"{cliente.nome} non ha abbastanza soldi")
                else:
                    print(f"{cliente.nome} ha pagato {incasso_tot} alla cassa
{self.numero}")
                    cliente.pagato = True # cliente ha pagato
                self.libera = True  # imposta cassa come libera
                break
```

Cassa ha come attributo il numero, in più nell'init inizializzo il mutex, utile perché così i clienti accedono esclusivamente ad una cassa, e la variabile bool libera, che mi serve per controllare se una cassa è libera oppure occupata. Il metodo test and set restituisce il valore precedente di libera e contemporaneamente la imposta come occupata. Questo è utile per verificare lo stato precedente di una variabile e, se necessario, modificarlo in modo atomico senza che altri thread intervengano. Il metodo paga, richiamato all'interno del run, permette ai clienti di pagare una volta finita la spesa. Faccio un controllo per vedere se la cassa scelta è libera, se lo è il cliente può continuare con il pagamento. Esso va a buon fine solo se il conto del cliente è sufficiente per pagare la sua spesa.

```
print("BENVENUTI AL SUPERMERCATO!!")
nome_supermercato = input("Inserisci il nome del supermercato: ")
indirizzo = input("Inserisci l'indirizzo: ")
città = input("Inserisci la città: ")
capienza = int(input("Inserisci la capienza massima del supermercato: "))
supermercato = Supermercato(nome_supermercato, indirizzo, città, capienza)
supermercato.carica_prodotti("Supermercato/magazzino.json") # carica prodotti dal
file scritto nel percorso

while(len(supermercato.clienti) < supermercato.capienza): # gira finchè la capienza
del supermercato non è raggiunta
    nome = input("Inserisci il nome del cliente: ")
    cognome = input("Inserisci il cognome del cliente: ")
    conto = float(input("Quanto ha nel conto corrente: "))
    cliente = Cliente(nome, cognome, conto) # creazione cliente
    cliente.entra() # cliente entra nel supermercato
    print("Vuoi aggiungere un altro cliente? (s/n)") # s = si, n = no
    agg = input()
    if ((agg == 'n')): # se l'utente non vuole aggiungere altri clienti, esco dal
while
        break
    elif len(supermercato.clienti) >= supermercato.capienza: # controllo capienza
        print("Il supermercato è pieno")
        break


for cliente in supermercato.clienti: # scorro clienti
    cliente.start() # avvio thread cliente
for cliente in supermercato.clienti: # scorro clienti
    cliente.join() # attendo la terminazione del thread cliente
for cliente in supermercato.clienti: # scorro clienti
    cliente.esce() # clienti escono dal supermercato
```

Questo è il main dove inizialmente viene creato, tramite inserimenti da tastiera, il supermercato e successivamente vengono carica i dati contenuti nel file .json. Successivamente troviamo la creazione dei clienti, anche questa fatta dall'utente da tastiera. Il ciclo si interrompe o quando la capienza del supermercato è stata raggiunta oppure quando l'utente nega la richiesta di inserimento di un ulteriore cliente. Infine troviamo l'avvio dei thread, tramite lo start, la join, ovvero l'attesa che tutti i thread finiscano la loro esecuzione, e successivamente un for per far uscire tutti i clienti dal supermercato.

RISULTATI E DIFFICOLTA’

BENVENUTI AL SUPERMERCATO!!
Inserisci il nome del supermercato: Famila
Inserisci l'indirizzo: Via Strada Solfanuccio
Inserisci la città: San Costanzo
Inserisci la capienza massima del supermercato: 5
Inserisci il nome del cliente: Mario
Inserisci il cognome del cliente: Rossi
Quanto ha nel conto corrente: 40
Mario è entrato nel supermercato
Vuoi aggiungere un altro cliente? (s/n)
s
Inserisci il nome del cliente: Luca
Inserisci il cognome del cliente: Verdi
Quanto ha nel conto corrente: 40
Luca è entrato nel supermercato
Vuoi aggiungere un altro cliente? (s/n)
s
Inserisci il nome del cliente: Marta
Inserisci il cognome del cliente: Talamelli
Quanto ha nel conto corrente: 50
Marta è entrato nel supermercato
Vuoi aggiungere un altro cliente? (s/n)
s
Inserisci il nome del cliente: Lucrezia
Inserisci il cognome del cliente: Vagabondi
Quanto ha nel conto corrente: 45
Lucrezia è entrato nel supermercato
Vuoi aggiungere un altro cliente? (s/n)
s
Inserisci il nome del cliente: Alessandro
Inserisci il cognome del cliente: Bastoni
Quanto ha nel conto corrente: 35
Alessandro è entrato nel supermercato
Vuoi aggiungere un altro cliente? (s/n)
n
Mario sta facendo la spesa
Mario sta comprando Shampoo
Quanti ne vuoi? 1
Luca sta facendo la spesa
Marta sta facendo la spesa
Marta non sta trovando il prodotto desiderato: Acqua
Per trovare Acqua vai al reparto: Prodotti Alimentari
Alessandro sta facendo la spesa
Alessandro non sta trovando il prodotto desiderato: Mela
Per trovare Mela vai al reparto: Frutta e Verdura
Mario sta comprando Gnocchi
Quanti ne vuoi? 1
Luca non sta trovando il prodotto desiderato: Cioccolato
Prodotto non trovato
Lucrezia sta facendo la spesa
Marta non sta trovando il prodotto desiderato: Patatine

```
Prodotto non trovato
Alessandro non sta trovando il prodotto desiderato: Prosciutto Crudo
Per trovare Prosciutto Crudo vai al reparto: Macelleria
Mario non sta trovando il prodotto desiderato: Lonza
Prodotto non trovato
Lucrezia non sta trovando il prodotto desiderato: Salame
Prodotto non trovato
Luca non sta trovando il prodotto desiderato: Yogurt
Per trovare Yogurt vai al reparto: Latticini
Marta non sta trovando il prodotto desiderato: Arancia
Per trovare Arancia vai al reparto: Frutta e Verdura
Alessandro ha finito di fare la spesa
Alessandro ha speso 0
Cassa 1 occupata
Mario ha finito di fare la spesa
Alessandro sta pagando 0 alla cassa 1
Mario ha comprato Shampoo
Mario ha comprato Gnocchi
Mario ha speso 3.88
Cassa 2 occupata
Mario sta pagando 3.88 alla cassa 2
Lucrezia sta comprando Farfalle
Quanti ne vuoi? 1
Luca ha finito di fare la spesa
Luca ha speso 0
Cassa 3 occupata
Marta ha finito di fare la spesa
Luca sta pagando 0 alla cassa 3
Marta ha speso 0
Lucrezia ha finito di fare la spesa
Lucrezia ha comprato Farfalle
Alessandro ha pagato 0 alla cassa 1
Lucrezia ha speso 0.89
Mario ha pagato 3.88 alla cassa 2
Cassa 2 occupata
Marta sta pagando 0 alla cassa 2
Luca ha pagato 0 alla cassa 3
Marta ha pagato 0 alla cassa 2
Cassa 2 occupata
Lucrezia sta pagando 0.89 alla cassa 2
Lucrezia ha pagato 0.89 alla cassa 2
Mario è uscito dal supermercato
Marta è uscito dal supermercato
Alessandro è uscito dal supermercato
PS C:\python>
```

Questo è il risultato del programma da terminale.
Scrivendo questo programma ho incontrato diverse difficoltà, come:

- Mancata conoscenza di come poter creare il file .xml per l'incasso partendo dal codice;
- Problemi per quanto riguarda l'accesso alle casse dei clienti;
- Problemi dovuti alla concorrenza dei thread.

Queste difficoltà le ho superate sia consultando materiale online che discutendone con i miei compagni.

CONCLUSIONI

Questo lavoro mi è stato utile sia da un punto di vista tecnico, perché molte delle cose che ho utilizzato nel programma non le avevamo mai affrontate a lezione e quindi sono andato a ricercare il loro significato e le loro applicazioni, sia da un punto di vista mentale, proprio perché senza questa assegnazione non avrei mai pensato di poter realizzare un programma che simulasse le spese dei clienti. Ho imparato nuove cose che riguardano Python e ho utilizzato anche file che prima non avevo mai visto come .json e .xml. Il progetto è stato stimolante e soprattutto non banale, poiché ci sono state alcune difficoltà nella scrittura del codice. Questo è sicuramente un aspetto positivo perché secondo me, sbattendoci la testa più volte, certe cose vengono comprese meglio e, un domani, in qualsiasi ambito, gli errori commessi saranno utili per cercare di non commetterli più.