II Prova in itinere di Programmazione Avanzata a.a. 2020-21

Istruzioni

Svolga gli esercizi nei file indicati e dopo aver concluso tutti gli esercizi comunichi alla docente di voler consegnare. Dopo aver ricevuto l'ok dalla docente mostri uno alla volta i file degli esercizi alla docente. Da quel momento non potra` piu` modificare il codice dell'esercizio e dovra` salvare tutti i file .py degli esercizi in una cartella zippata il cui nome deve essere formato dal suo cognome e nome separati da un '_' (ad esempio, rossi_mario). Invii la cartella zippata a adebonis@unisa.it con oggetto "Soluzione esercizi appello dicembre".

Durante lo svolgimento della prova, eccezion fatta per il collegamento al meeting di Zoom, è vietato l'uso di internet (navigazione tramite browser, posta, ssh, ftp, accesso remoto al PC, condivisione dello schermo e di qualsiasi cartella, ecc.).
Teams e Zoom devono essere ridotti ad icona.

NON MODIFICHI IL CODICE GIA` SCRITTO NEI FILE FORNITI DALLA DOCENTE (0 punti se cio` viene fatto)

1. Scrivere una classe Alimento che, oltre al nome, ha i seguenti attributi che ne determinano lo stato**:**

   - **scorta** : quantita` di scorte dell'alimento presente nella cucina di un ristorante del (inizialmente e` 0)
   - **disponibilita** che puo` avere tre valori: "elevata", "media", "nessuna" (inizialmente e` "nessuna").
   - **_consumo**: dizionario che tiene traccia del consumo durante l'anno dell'alimento per cucinare le pietanze del menu. Le entrate hanno come chiave il nome di una pietanza e come valore una tupla di due elementi (quantita, data) dove quantita e` la quantita` di alimento richiesta per cucinare la pietanza fino a quel giorno e viene aggiornata a fine giornata se qualche cliente ha ordinato quella pietanza, e data e` la data in cui e` avvenuto l'ultimo aggiornamento di quantita.

La classe ha anche due metodi _usa , _fai_scorta e butta_scorte, forniti per vostra convenienza dalla docente. La classe deve anche fornire le versioni pubbliche usa e fai_scorta dei metodi _usa e fai_scorta, rispettivamente.

Le variabili MAXIMM e MAXCONSUMO (anch'esse gia` fornite) contengono il massimo quantitativo di alimento che il ristorante puo` immagazzinare e il massimo quantitativo di alimento che si puo` consumare per cucinare una pietanza.

- Gli attributi scorta e disponibilita` scorta e disponibilita sono accessibili con il loro nome e modificabili con '=' mentre dizionario _consumo e` modificabile con il seguente metodo:
  - il metodo **aggiorna(self,nome,usata,nuovadata)** che prende in input il nome di una pietanza, la quantita` usata quel giorno per preparare la pietanza e la data di quel giorno. Il metodo aggiorna la coppia (quantita, data) associata a nome con la coppia (quantita+usata,nuovadata).

- **Usare l'approccio state specific per implementare disponibilita` tenendo conto che se disponibilita è "elevata" non e` possibile immagazzinare ulteriori quantita` dell'alimento mentre se e` "nessuna" allora non e` possibile usare l'alimento. Zero punti a chi non usa questo approccio.**
- **Il nome del alimento e` una stringa settata da __init__ (self,nome).**

Per quelli che sono sicurissimi di saper svolgere correttamente l'esercizio 2: potete usare nell'esercizio 1 direttamente la classe AlimentoOsservato (importandola) e definendo poi la classe alimento in questo modo *class Alimento(AlimentoOsservato): pass* .

2. Scrivere la classe AlimentoOsservato che include gli stessi attributi e gli stessi metodi della classe Alimento ma che, a differenza di Alimento, puo` essere osservata da uno o piu` osservatori. Scrivere inoltre i seguenti due osservatori di AlimentoOsservato:

   - **Ristorante** che si comporta nel modo seguente.

   a) Se viene cucinata una pietanza che richiede l'uso del alimento,
      - stampa "\nNuovo uso dell'alimento {}: quantita` disponibile = {}", dove, al posto delle parentesi devono comparire il nome dell'alimento e la quantita` di alimento disponibile.
      - Inoltre, se in seguito all'uso del alimento, la quantita` disponibile e` diventata minore di 1/4 di MAXIMM, allora immagazzina una quantita` di alimento pari alla differenza tra MAXIMM e la quantita` di alimento disponibile. Prima di far questo, stampa "Necessario fare scorta dell'alimento {}", dove al posto delle parentesi deve comparire il nome del alimento.

   b) Se e` stata immagazzinata un nuovo quantita` di alimento:
      - stampa "Immagazzinata una nuova quantita` dell'alimento {}: quantita` disponibile = {}". Al posto delle parentesi devono comparire il nome dell'alimento e la quantita` disponibile.

   c) Se e` stato effettuato un nuovo uso per cucinare una pietanza che non era presente nel dizionario _consumo, stampa "Alimento {} usato per cucinare una pietanza mai cucinata dall'inizio dell'anno". Al posto delle parentesi, deve comparire il nome del alimento.

   d) Se è cambiato il valore di disponibilita, stampa "È cambiata la disponibilita` dell'alimento {}: la disponibilita` ora è {} ", dove al posto delle parentesi devono comparire il nome e il valore di disponibilita.

   - **StoricoRistorante** che mantiene una lista di stringhe **storico** aggiornandola nel modo seguente:
      - aggiunge alla lista \nGiorno {}: l'alimento {} è stato usato per cucinare una pietanza mai cucinata da inizio anno", nel caso in cui l'alimento sia stato usato per preparare una pietanza che non era gia` presente nel dizionario _consumo. Al posto delle parentesi deve comparire la data in cui è avvenuto l'uso dell'alimento e il nome dell'alimento.

   NON mantenete informazioni sull'oggetto/oggetti osservati negli osservatori.

3. Scrivere una diversa versione dell'observer Ristorante dell'esercizio 2 in cui il metodo update di utilizza una catena di 5 gestori: un gestore per ciascuno dei 4 casi a), b), c), d) dell'esercizio 2, piu` un gestore di default che stampa un newline.
   Al caso **a)** e` associato il **gestore gestore_uso**, al caso **b)** il gestore **gestore_nuova_scorta**, al caso **c)** il gestore **gestore_np**, e infine al caso **d)** il **gestore_ cambioStato**.
   Gli studenti che non sono stati in grado di scrivere il codice dell'esercizio 2 possono usare al posto del file esercizio3.py il file **esercizio3DIRISERVA.py** che contiene una classe che **simula** il comportamento di quella descritta nell'esercizio2.

4. Scrivere la funzione processaProdotti all'interno del file esercizio4.py. Se la funzione ha bisogno di invocare altre procedure, fornire anche queste ultime. La funzione processaAlimento prende in input, oltre al parametro concorrenza che indica il numero di processi in uso, una lista **alimenti** di alimenti (oggetti di tipo AlimentoSemplice) e una lista **liste_pietanze** di liste di triple della forma (nomePietanza, quantita,data), dove nomePietanza e` il nome della pietanza, quantita è la quantita` di alimento usata per cucinare la pietanza e data è la data in cui viene usata quella quantita` dell'alimento (e` un oggetto di tipo datetime.date).
   Facendo uso di **Futures** oppure di **JoinableQueue**, la funzione processaAlimenti deve aggiungere al dizionario _consumo dell'i-esimo alimento della lista **alimenti,** le pietanze (con le relative informazioni) presenti nell'i-esima lista di **liste_pietanze**. Cio` deve essere fatto in modo concorrente (piu` processi che allo stesso tempo si occupano ciascuno di un alimento diverso). La funzione processaAlimenti deve inoltre stampare per ciascun alimento il numero di pietanze dopo i nuovi inserimenti e tutte le entrate del dizionario _consumo, **incluse quelle presenti in precedenza nel dizionario**. **Le stampe devono avvenire nell'ordine in cui terminano i processi.**
   La classe AlimentoSemplice e` gia` presente nel file.