Tema 1 - Marketplace

- **Deadline:** 11 aprilie 2022, ora 23:55. Primiți un bonus de 10% pentru trimiterea temei cu 3 zile înaintea acestui termen, adică înainte de 8 aprilie 2021, ora 23:55.
- **Deadline hard:** 18 aprilie 2022, ora 23:55. Veți primi o depunctare de 10% din punctajul maxim al temei pentru fiecare zi de întârziere, până la maxim 7 zile, adică până pe 18 aprilie 2022, ora 23:55.
- **Responsabili:** Voichița Iancu [mailto:voichita.iancu@cs.pub.ro], Eduard Stăniloiu [mailto:eduard.staniloiu@cs.pub.ro], Giorgiana Vlăsceanu [mailto:giorgiana.vlasceanu@gmail.com]
- **Autori:** Luca Istrate [mailto:LucaIstrate@gmail.com], Adriana Draghici [mailto:adriana.draghici@cs.pub.ro], Loredana Soare [mailto:soareloredana97@gmail.com], Eduard Stăniloiu [mailto:eduard.staniloiu@cs.pub.ro]

- Dată publicare: 28 martie
- Dată actualizare enunț: 28 martie

Scopul temei

- Utilizarea eficientă a elementelor de sincronizare studiate la laborator
- Implementarea unei aplicații concurente utilizând o problemă clasică (Multi Producer, Multi Consumer)
- Aprofundarea anumitor elemente din Python (clase, elemente de sintaxă, thread-uri, sincronizare, precum și folosirea modulelor Python pentru lucrul cu thread-uri)

Enunț

În cadrul acestei teme veți avea de implementat un Marketplace prin intermediul căruia mai mulți **producători** își vor oferi produsele spre vânzare, iar mai mulți **cumpărători** vor achiziționa produsele puse la dispoziție.

Marketplace

Marketplace-ul este unul destul de simplu, cu **două tipuri de produse (ceai și cafea)** ce vor fi comercializate de către producători. Acesta va fi intermediarul dintre producători și consumatori, prin el realizându-se achiziția de produse: producătorul (producer) va produce o anumită cantitate de produse de un anumit tip / mai multe tipuri cumpărătorul (consumer) va cumpăra o anumită cantitate de produse de un tip / de mai multe tipuri. De asemenea, Marketplace-ul va pune la dispoziția fiecărui cumpărător câte un **coș de produse (cart)** (acesta va fi folosit pentru rezervarea produselor care se doresc a fi cumpărate).

Producător

Vor exista mai mulți producători ce vor produce obiectele de tip cafea / ceai. Fiecare produs va fi furnizat într-o anumită cantitate. Un producător poate produce atât obiecte de tip cafea, cât și de tip ceai.

Consumator

În momentul în care un client își dorește să cumpere anumite produse dintr-un magazin, acesta va avea nevoie de un coș de cumpărături pe care să îl folosească în scopul rezervării acestora. Astfel, de fiecare dată când un client își începe cumpărăturile, acesta va primi din partea Marketplace-ului un coș de cumpărături, căruia îi va fi asociat un *id*. Clientul poate:

- adăuga produse în coș ⇒ produsele respective devin indisponibile pentru ceilalți clienți
- șterge produse din coș ⇒ produsele respective devin disponibile pentru ceilalți clienți
- plasa o comandă

Descrierea implementării

Marketplace-ul ce va trebui implementat va simula problema **Multi Producer Multi Consumer (MPMC)**. Pentru rezolvarea acestei teme va trebui să completați clasele `Marketplace`, `Producer`, și `Consumer` cu o implementare corectă a metodelor deja definite.

Rezolvarea temei va fi concentrată preponderent pe metodele clasei *Marketplace*, metode ce vor fi apelate atât de producător, cât și de cumpărător în clasele aferente ale acestora.

Operația efectuată de către producător este cea de *publicare a produselor sale*. Implementarea metodei `publish` va fi făcută în clasa *Marketplace*.

Vor exista doua tipuri de operații pe care clientul le poate efectua asupra coșului de cumpărături:

- `add_to_cart` ⇒ adaugă un produs în coș
- `remove_from_cart` ⇒ șterge un produs din coș

Ambele metode (`add_to_cart` și `remove_from_cart`) vor trebui implementate în clasa *Marketplace*.

În momentul în care un consumator adaugă un produs în coșul pentru cumpărături, produsul respectiv va deveni indisponibil pentru ceilalți clienți ai Marketplace-ului. Clientul își va putea plasa comanda prin apelarea metodei `place_order` (din clasa Marketplace). În cazul în care un produs este eliminat din coșul pentru cumpărături, acesta devine disponibil pentru ceilalți clienți ai Marketplace-ului.

Funcționalitatea clasei `Producer` este să:

- furnizeze produselor pe care producătorul le pune la dispoziție

`Producer` produce secvențial numărul de produse și tipul din cadrul fișierului de intrare și așteaptă după realizarea fiecărui produs un număr de secunde specificat. Informațiile se preiau din fișierul de intrare și are următorul format pentru produse`[“id”, cantitate, timp-așteptare]`.

Funcționalitatea clasei `Consumer` este să:

- primească id-ului coșului de cumpărături
- adauge / elimine din coșul de cumpărături anumite cantități de produse
- plaseze comenzi

Modulul **Product** conține reprezentările claselor **Coffee** și **Tea**.

Marketplace-ul limitează numărul de produse ce pot fi publicate de către un producător. În momentul în care s-a atins limita, producătorul nu mai poate publica altele până nu sunt cumpărate. El va reîncerca să publice după un timp definit în fișierul de test.

Dacă un cumpărător nu găsește un produs în marketplace, el va încerca mai târziu, după un timp definit în fișierul de test.

Se consideră timp de așteptare după:

- adăugarea unui produs
- semnalizarea că nu se găsește un produs
- semnalizarea faptul că este plină coada asociată producătorului

Testare

Testarea se va realiza folosind atât `unitteste`, cât și teste funcționale.

Unittesting

Pentru testarea funcțiilor din `Marketplace` veți folosi modulul de unittesting [https://docs.python.org/3/library/unittest.html] al limbajului Python.

Pentru a defini un set de unitteste trebuie să vă definiți o clasă care moștenește clasa `unittest.TestCase`

demo_unittest.py

```
import unittest

class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')
```

Pentru a defini un test, numele metodei trebuie să înceapă cu prefixul `test_`, așa cum puteți observa în exemplul de mai sus: `test_upper`. Verificările din corpul metodei se fac folosind metodele `assert*`, în exemplul de mai sus a fost folosită metoda `assertEqual`. O listă completă a metodelor de verificare disponibile este prezentată în documentație [https://docs.python.org/3/library/unittest.html#assert-methods].

Pentru a rula testele, folosim subcomanda unittest:

```
$ python3 -m unittest demo_unittest.py
$ # puteti folosi optiunea -v pentru mai multe detalii
$ python3 -m unittest -v demo_unittest.py
```

Pentru a testa comportamentul clasei `Marketplace` definiți în fișierul `marketplace.py` o clasă de testare numită `TestMarketplace`. Clasa `TestMarketplace` va testa funcționalitatea tuturor metodelor definite de `Marketplace`: `register_producer`, `publish`, `new_cart`, `add_to_cart`, `remove_from_cart`, `place_order`. Dacă definiți alte metode, va trebui să adăugați teste și pentru acestea.

Vă recomandăm să folosiți metoda setUp [https://docs.python.org/3/library/unittest.html#unittest.TestCase.setUp] pentru a inițializa o instanță a clasei testate (`Marketplace`) și orice altceva ce vă ajută în testarea codului. Un exemplu de utilizare a metodei `setUp` este disponibil în documentație [https://docs.python.org/3/library/unittest.html#organizing-test-code].

Testarea Funcțională și Formatul Testelor

Testarea se va face cu ajutorul a două tipuri de fișiere, cele de input și cele de output ({id}.in și {id}.out), primul fiind în format JSON. Fișierul **{id}.in** va reprezenta fișierul de intrare și va conține configurările necesare pentru fiecare clasă în parte, iar fișierul **{id}.out** va reprezenta fișierul de ieșire prin intermediul căruia se va verifica corectitudinea implementării temei.

Fișierele de input vor fi fișiere JSON ce vor conține următoarele chei:

- marketplace
- products
- producers

- consumers

Exemplu conținut fișier de intrare și fișierul corespunzător de ieșire:

```
{
   "products": {
       "id1": {
           "product_type": "Coffee",
           "name": "Arabica",
           "price": 10,
           "acidity": 5.1,
           "roast_level": "medium"
       },
       "id2": {
           "product_type": "Tea",
           "name": "Earl Grey",
           "price": 10,
           "type": "Green"
       }
   },
   "consumers": [
       {
           "name": "cons1",
           "retry_wait_time": 0.1,
           "carts": [
               [
                   { "type": "add", "prod": "id1", "qty": 2 },
                   { "type": "remove", "prod": "id1", "qty": 1 }
               ],
               [
                   { "type": "add", "prod": "id2", "qty": 3 }
               ]
           ]
       }
   ],
   "producers": [
       {
           "name": "prod1",
           "products": [
               [ "id1", 1, 0.1 ],
               [ "id2", 1, 0.1 ]
           ],
           "republish_wait_time": 0.2
       },
       {
           "name": "prod2",
           "products": [
               [ "id2", 1, 0.2 ]
           ],
           "republish_wait_time": 0.2
       }
   ],
   "marketplace": {
       "queue_size": 8
   }
}
```

Conținut fișier de ieșire:

```
cons1 bought Coffee(name='Arabica', price=10, acidity=5.1, roast_level='medium')
cons1 bought Tea(name='Earl Grey', price=10, type='Black')
cons1 bought Tea(name='Earl Grey', price=10, type='Black')
cons1 bought Tea(name='Earl Grey', price=10, type='Black')
```

Atât conținutul fișierului de intrare, cât și conținutul fișierului de ieșire sunt descrise în README [https://gitlab.cs.pub.ro/asc/asc-public/-/blob/master/assignments/1-marketplace/skel/test-gen/README_TESTS.md]

Pentru a putea compara fișierele de ieșire obținute de voi cu cele de referința, scriptul de testare va ordona output-ul rezultat, întrucât avem de-a face cu multithreading.

Logging

Vrem să utilizăm fișiere de logging în aplicațiile pe care le dezvoltăm pentru a putea urmări flowul acestora a.î. să ne ajute în procesul de debug.

Folosind modulul de logging [https://docs.python.org/3/library/logging.html], trebuie să implementați un fișier de log, numit "marketplace.log", în care veți urmări comportamentul clasei `Marketplace`.

În fișierul de log veți nota, folosind nivelul `info()`, toate intrările și ieșirile în/din metodele clasei `Marketplace`. În cazul metodelor care au parametrii de intrare, informația afișată la intrarea în funcție va afișa și valorile parametrilor. Fișierul va fi implementat folosind RotatingFileHandler [https://docs.python.org/3/library/logging.handlers.html#logging.handlers.RotatingFileHandler]: astfel se poate specifica o dimensiune maximă a fișierului de log și un număr maxim de copii istorice. RotatingFileHandler ne permite să ținem un istoric al logurilor, fișierele fiind stocate sub forma "file.log", "file.log.1", "file.log.2", ... "file.log.max".

Vă încurajăm să folosiți fișierul de log și pentru a înregistra erori [https://docs.python.org/3/library/logging.html#logging.Logger.error] detectate.

În mod implicit, timestamp-ul logurilor folosește timpul mașinii pe care rulează aplicația (local time). Acest lucru nu este de dorit în practică deoarece nu putem compara loguri de pe mașini aflate în zone geografice diferite. Din acest motiv, timestampul este ținut în format UTC/GMT. Asigurați-vă că folosiți gmtime, și nu localtime. Pentru aceasta trebuie să folosiți metoda formatTime [https://docs.python.org/3/library/logging.html#logging.Formatter.formatTime].

O descriere completă a cum puteți utiliza modulului de logging este prezentă în categoria HOWTO [https://docs.python.org/3/howto/logging.html] a documentației.

Precizări încărcare / VMChecker

Arhiva temei va fi încărcată pe vmchecker [https://vmchecker.cs.pub.ro/ui/#ASC].

Arhiva trebuie să conțină:

- director `tema` cu fișierele temei: `marketplace.py`, `producer.py`, `consumer.py`
- alte fișiere `.py` folosite în dezvoltare
- `README`
- director .git

Pentru a documenta realizarea temei, vă recomandăm să folosiți template-ul de aici [https://gitlab.cs.pub.ro/asc/asc-public/-/blob/master/assignments/README.example.md]

Punctare

Tema va fi verificată automat, folosind infrastructura de testare, pe baza unor teste definite în directorul `tests`.

Tema se va implementa **Python>=3.7**.

Notarea va consta în 80 pct acordate egale între testele funcționale, 10 pct acordate pentru unitteste și 10 pct acordate pentru fișierul de logging. Depunctări posibile sunt:

- folosirea incorectă a variabilelor de sincronizare (ex: lock care nu protejează toate accesele la o variabilă partajată, notificări care se pot pierde) (-2 pct)
- prezența print-urilor de debug (maxim -10 pct în funcție de gravitate)
- folosirea lock-urilor globale (-10 pct)

- folosirea variabilelor globale/statice (-5 pct)
    - Variabilele statice pot fi folosite doar pentru constante
- folosirea inutilă a variabilelor de sincronizare (ex: se protejează operații care sunt deja thread-safe) (-5 pct)
- alte ineficiențe (ex: creare obiecte inutile, alocare obiecte mai mari decât e necesar, etc.) (-5 pct)
- lipsa organizării codului, implementare încâlcită și nemodulară, cod duplicat, funcții foarte lungi (între -1pct și -5 pct în funcție de gravitate)
- cod înghesuit/ilizibil, inconsistenţa stilului - vedeți secțiunea Pylint
    - pentru code-style recomandăm ghidul oficial PEP-8 [https://www.python.org/dev/peps/pep-0008/]
- cod comentat/nefolosit (-1 pct)
- lipsa comentariilor utile din cod (-5 pct)
- fişier README sumar (până la -5 pct)
- nerespectarea formatului .zip al arhivei (-2 pct)
- alte situaţii nespecificate, dar considerate inadecvate având în vedere obiectivele temei; în special situațiile de modificare a interfeței oferite

Se acordă bonus 5 pct pentru adăugarea directorului `.git` și utilizarea versionării în cadrul repository-ului.

Temele vor fi testate împotriva plagiatului. Orice tentativă de copiere va fi depunctată conform regulamentului.

Pylint

Vom testa sursele voastre cu pylint [https://www.pylint.org/] configurat conform fișierului **`pylintrc`** din cadrul repo-ului dedicat temei. Atenție, rulăm pylint doar pe modulele completate și adăugate de voi, nu și pe cele ale testerului.

Deoarece apar diferențe de scor între versiuni diferite de pylint, vom testa temele doar cu ultima versiune [https://www.pylint.org/#install]. Vă recomandăm să o folosiți și voi tot pe aceasta.

Vom face depunctări de până la -5pct dacă verificarea făcută cu pylint vă dă un scor mai mic de 8.

Observații

- Pot exista depunctări mai mari decât este specificat în secţiunea Notare pentru implementări care nu respectă obiectivele temei și pentru situatii care nu sunt acoperite în mod automat de către sistemul de testare
- Implementarea şi folosirea metodelor oferite în schelet este obligatorie
- Puteți adăuga variabile/metode/clase, însă nu puteți schimba antetul metodelor oferite în schelet
- Bug-urile de sincronizare, prin natura lor sunt nedeterministe; o temă care conţine astfel de bug-uri poate obţine punctaje diferite la rulări succesive; în acest caz punctajul temei va fi cel dat de tester în momentul corectării
- Recomandăm testarea temei în cât mai multe situații de load al sistemului și pe cât mai multe sisteme pentru a descoperi bug-urile de sincronizare

Resurse necesare realizării temei

Pentru a clona repo-ul [https://gitlab.cs.pub.ro/asc/asc-public] și a accesa resursele temei 1:

```
student@asc:~$ git clone https://gitlab.cs.pub.ro/asc/asc-public.git
student@asc:~$ cd asc/assignments
```

```
student@asc:~/assignments$ cd 1-marketplace
```

Suport, întrebări și clarificări

Pentru întrebări sau nelămuriri legate de temă folosiți forumul temei [https://curs.upb.ro/2021/mod/forum/view.php?id=182065].

Orice intrebare e recomandat să conțină o descriere cât mai clară a eventualei probleme. Întrebări de forma: "Nu merge X. De ce?" fără o descriere mai amănunțită vor primi un răspuns mai greu.

ATENȚIE să nu postați imagini cu părți din soluția voastră pe forumul pus la dispoziție sau orice alt canal public de comunicație. Dacă veți face acest lucru, vă asumați răspunderea dacă veți primi copiat pe temă.