DOCUMENTATIE
TEMA NR. 2

Mudura Ana-Andreea

Grupa 30221

CUPRINS

1.Obiectivul temei

Obiectivul temei este realizarea unei aplicatii de gestionare a a cozilor utilizand threadurile si mecanisme de sincronizare / structuri de date sigure in utilizarea sincronizarii.

Se va discuta in capitolele urmatoare

- Analiza problemei: cazurile functionale,diagrame,use-case-urile temei
- Proiectare: se va descrie aplicatia din punct de vedere a principiilor de programare orientata pe obiecte cu clase,pachete,intefrete si algoritmi folositi
- Implementare:se va descrie detaliat fiecare clasa in parte si interfata dedicata utilizatorului
- Rezultate: rezultatele obtinute in fisierele dedicate pentru retinerea in timp real a evenimentelor aplicatiei
- Concluzii: cunostiinte acumulate,imbunatatiri pe viitor
- Bibliografie

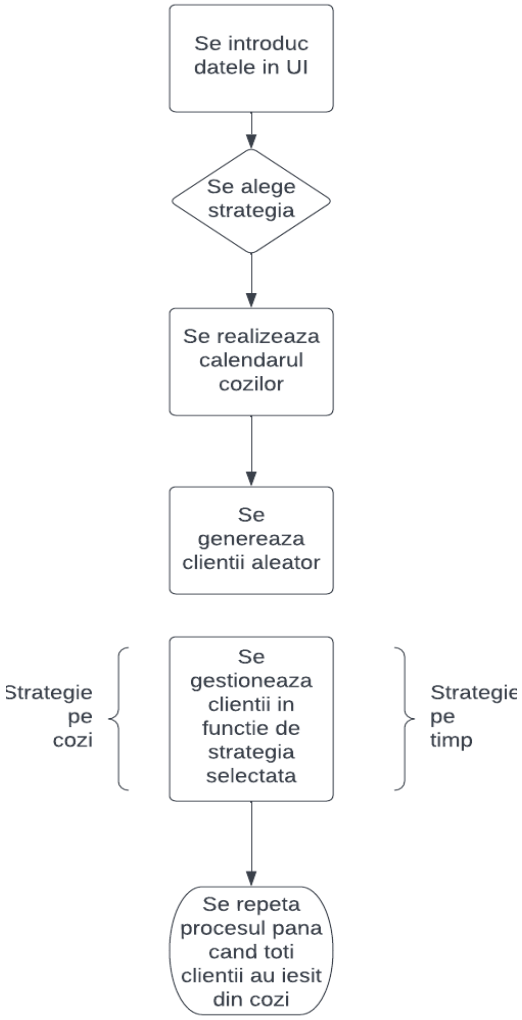
2.Analiza problemei

Aplicatia de gestionare a cozilor prin intermediul threadurilor si a mecansismelor de functionare are ca si date de intrare urmatoarele informatii

- Numarul de clienti
- Numarul de cozi disponibile clientilor
- Numarul de secunde care reprezinta durata simularii
- Numarul minim si maxim de ajungere a unui client pe durata simuarii(nr minim <=nr maxim)
- Numarul minim si maxim de procesare a unui client pe durata simularii(nr minim<=nr maxim)

Problema realizarii aplicatiei consta in impartirea sa in diferite etape(dupa cum se poate observa si din diagrama). Pentru fiecare client se obtine un id, un timp minim/maxim de ajungere, un timp maxim/minim de procesare. Pentru fiecare coada in parte(numarul maxim de cozi este obtinut din interfata grafica) se creeaza un thread, la fel si pentru simulare. Pentru fiecare sever in parte(care reprezinta coziile) se intocmeste un Scheduler, care atribuie fiecarei cozi clienti in functie de strategia selectata. Strategiile pot sa fie in functie de cozi(clientul se adauga in coada cea mai scurta) sau in functie de timpul de asteptare calculat in timp real(clientul se adauga in coada care are timpul de asteptare cel mai mic). Simularea se termina in momentul in care nu mai exista nici un client in vreo coada deschisa si nu mai exista nici un client in zona de asteptare. Toate informatiile simularii sunt transmise intefetei de simulare in timp real si unui fisier text deschis la lansarea threadului simularii.

In flow chart-ul de mai jos este exemplificat use-case-ul aplicatiei, pasii urmand sa fie mai detaliati in capitolele ce urmeaza



3.Proiectare

In aplicatie sunt folosite unele dintre paragdigmele de programare orientata pe obiecte: mostenirea si incapsularea.

Încapsularea (care mai poate fi interpretata ca si ascunderea de informații) asigura faptul ca obiectele încapsulate nu pot schimba starea interna a altor obiecte in mod direct(accesul se poate realiza doar prin intermediul metodelor proprii). In alta ordine de cuvinte, starea obiectului curent poate fi schimbata doar de metodele puse la dispoziție de obiectul respectiv.

Pentru "ascunderea informatiilor", toate datele introduse din interfata care apoi sunt folosite de catre program sunt private, inclusiv serverele, organizatorul, datele clientului care sunt generate aleator si interfetele grafice pentru manager si afisarea simularii in timp real.

Pentru realizarea gestionarii cozilor in cadrul aplicatiei este folosit multithreadingul, care este o caracteristică Java care permite execuția concomitentă a două sau mai multe părți ale unui program pentru utilizarea maximă a procesorului. Fiecare parte a unui astfel de program se numește fir. Deci, firele sunt procese ușoare în cadrul unui proces. Beneficiile acestei tactici le reprezinta ultilizarea procesorului cum am mentionat mai sus,minimizarea utilizarii resurselor,structura simplificata si comunicarea imbunatatia intre procese.

In diagrama UML următoare se pot observa relațiile dintre clase împreuna cu pachetele folosite si interfețele utilizate.

Intre Client si Server exista o relatie de dependenta, aceasta relatie este intrepretata: fiecare server reprezinta o coada de clienti(la noi este implementata cu ajutorul unei BlockingQueue) cu un numar maxim de locuri citit de la interfata grafica. Schimbarea unuia dintre elemente poate cauza o schumbare in celalalt. Totodata exista o relatie de mostneire Client-Server deoarece se mai poate interpreta si fiecare Servers este o coada de clienti, deci afirmatia se dovedeste fiind adevarata.

Intre Server si Scheduler exista o relatie de asociere ca si agregare, adica organizatorul este o parte a severului pentru ca el contine strategiile de distribuire a clientilor. Totodata exista si relatia de mostenire deoarece se mai poate interpreta ca si Serverul sa fie un Scheduler din moment ce fiecare componenta are un organizator. Prin generalizare, relatia de mostenrie se extinde si intre Scheduler si Client.

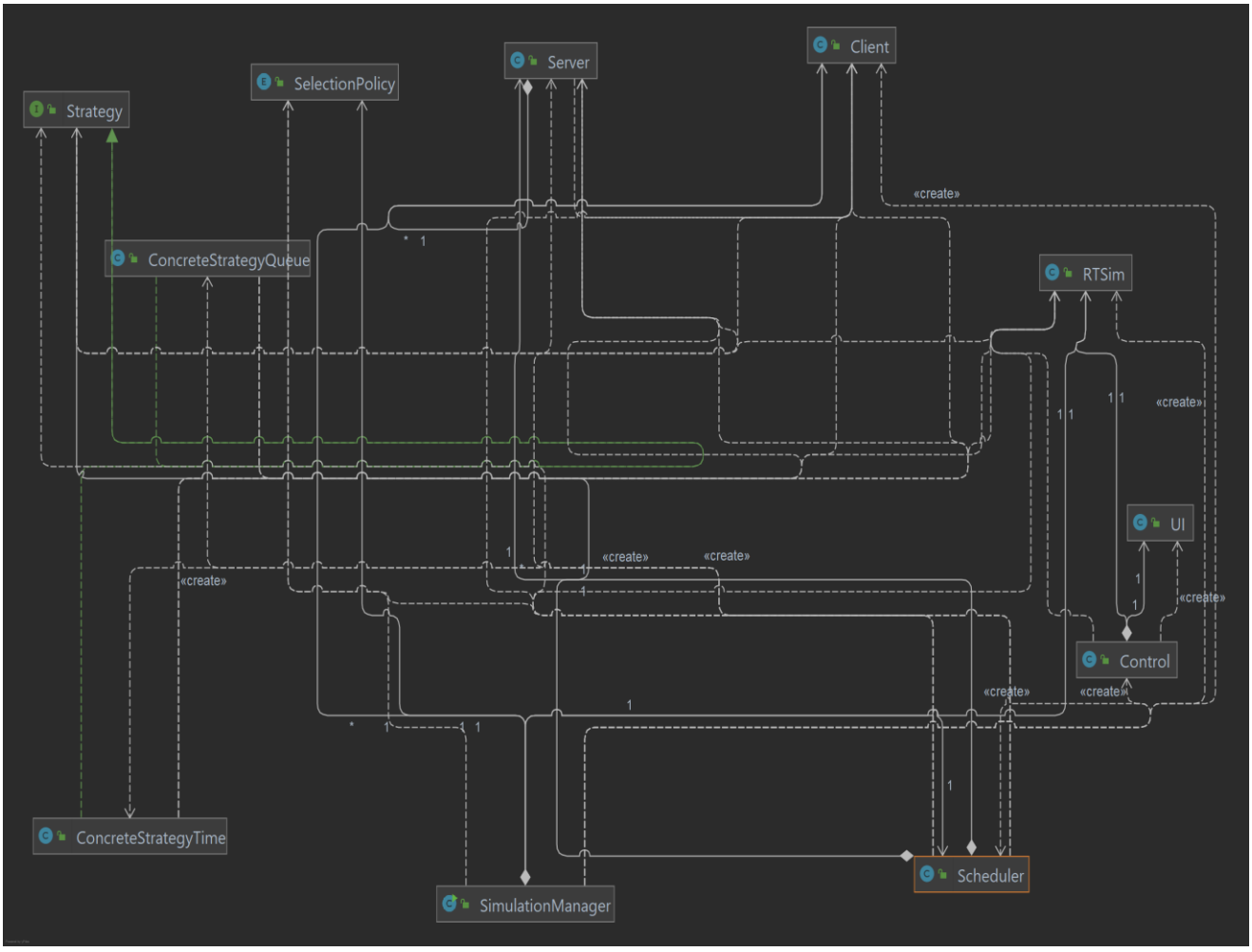
Relatiile dintre Control si celelalte clase(clasa principala care controleaza simularea) este predominant de dependenta. Clasa contine si organizatoarele,serverele, interfata de extragere a datelor si simulatorul aplicatiei. In simulatorul aplicatiei se regasesc datele extrase si interfata de afisare a simularii in timp real, intre care exista o relatie de generalizare.

Interfata implementata(Strategy) contine metoda de adaugare clienti care ii trimite in cozi in functia de selectia aleasa(dupa timpul cel mai scurt sau dupa coada cea mai scurta). Clasele care implementeaza interfata sunt ConcreteStrategyTime si ConcreteStrategyQueue care, in functie de SelectionPolicy, contin metoda de adaugare a clientului in coada din Scheduler.

Interfațe predefinite folosite in proiectarea aplicatiei suntActionListener, o interfața publica care primește ca argument un tip de date numit eveniment si exercita o acțiune(prin metoda

actionPerformed) asupra unui obiect de tip component, dedicat interfețelor grafice. In proiect, fiecare buton deține metoda addListener(Start– care la apăsarea lui se efectuează porneste simularea si deschide interfata grafica dedicata simularii in timp real – si Back– butonul de revenire la starea inițială a interfeței) ,interfata Comparable a colectiilor care ajuta la sortarea unei structure de date in functie de parametrii alesi de catre utilizator, si interfata Runnable a threadurilor care contine metoda run de pornire a firelor de lucru.

Nu s-au folosit algoritmi specifici in realizarea aplicatiei, metodele care ies in evidenta fiind algoritmii de generare a timpului de ajungere,de procesare si a id –ului aleator, parsarea intre intreg si string de la citirea din interfata grafica, schimbarea strategiilor si metodele de run si sleep specifica threadurilor.



4.Implementare

Implementarea aplicatiei a fost realizata cu un design de programare orietntata pe obiecte, folosind conventiile Java de denumire si folosind structuri de date adecvate pentru sincronizarea multithreadingului.

Au fost folosite metode stateless, BlockingArrayQueue(gestioneaza dimensiunea cozilor) pentru thread safety

Client

Clasa client are ca si campuri id-ul, timpul de ajungere si timpul de procesare/servire. Toate atributele clientului sunt generate aleator cu ajutorul metodelor specifice. Clasa implementeaza si interfata Comparable a colectiilor, drept urmare exista si metoda compareTo care sta la dispozitia sortarii listei de clienti generati in functie de timpul de ajungere.

Server

Clasa server(fiecare reprezinta o coada) are ca si campuri lista de clienti, timpul de asteptare a cozii si numarul maxim de elemnte din ea. Clasa implementeaza si interfata Runnable care implementeaza metoda run in care fiecare client este scos din coada,threadul este adormit, iar timpul de asteptare este scazut. Aceasta actiune este interpretata in mod real ca si clientul din coada este servit, drept urmare el nu mai trebuie sa astepte pentru un serviciu. Clasa mai are si metoda de adaugare client care incrementeaza timpul de asteptare(un client a ajuns la coada, deci timpul de asteptare al ei creste).

Scheduler

Clasa scheduler are ca si campuri lista de servere(lista de cozi), numarul maxim de clienti din cozi ,numarul maxim de cozi existente si strategia folosita in organizare. In constructor se creeaza o lista noua de cozi, iar pentru fiecare coada se lanseaza un thread. Clasa contine metoda de schimbare a strategiei(Time sau Queue) si metoda strategiei de adaugare client in functie de SelectionPolicy(metoda de adaugare in functie de timp calculeaza timpul minim al fiecuri sever si acolo adauga clientul; metoda de adaugare in functie de queue calculeaza cea mai scurta coada si acolo adauga clientul).

ConcreteStrategyTime si ConcreteStrategyQueue + interfata Strategy

Dupa cum am mentionat mai sus, cele doua clase contin metodele de adaugare a clientilor in functie de strategia aleasa(shortest time sau shortest queue).

Control

Clasa de control controleaza toata simularea incepand de la interfata grafica dedicata utilizatorului care extrage datele. Clasa lucreaza concomitent cu clasa SimulationManager care implementeaza interfata Runnable si are metoda de start a threadurilor. Controlul detine cele doua metode de ActionPerformed a butoanelor care pornesc sau opresc simularea. In metoda de start se extrag datele, pentru fiecare server se porneste un thread, se selecteaza strategia care urmeaza sa fie folosita, se intocmeste organiatorul in functie de numarul maxim de cozi,numarul maximi de clienti din cozi si strategie,se genereaza clienti aleatori si se porneste threadul simularii. Interfata grafica de extragere se inchide si interfata de afisare a datelor in timp real este deschisa. Toate datele sunt transmise Managerelui de simulare.

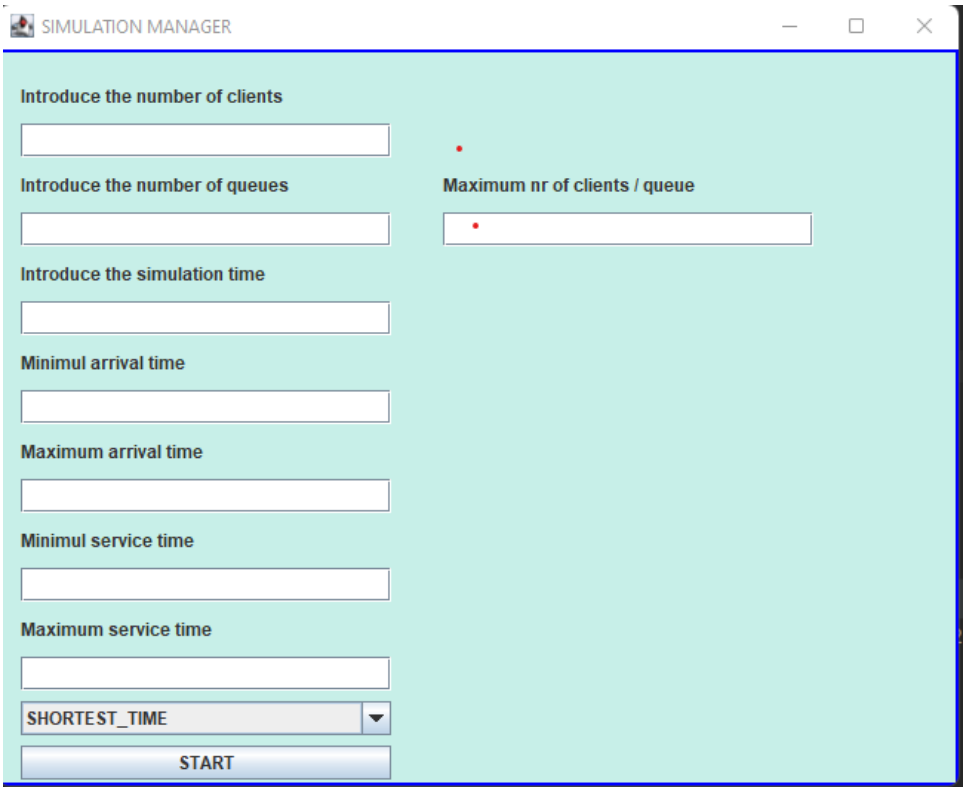
SimulationManager

Simulatorul aplicatiei contine toate datele extrase din interfata grafica de extragere incapsulate, lista de clienti generati aleatori si interfata grafica de afisare a cozilor si a clientilor aflati in asteptare in timp real. Metoda de generare a clientilor se foloseste de metodele fiecarui atribuit a clientilor apelate cu setteri si getteri si care au parametrii de intrare informatiile extrase anterior.

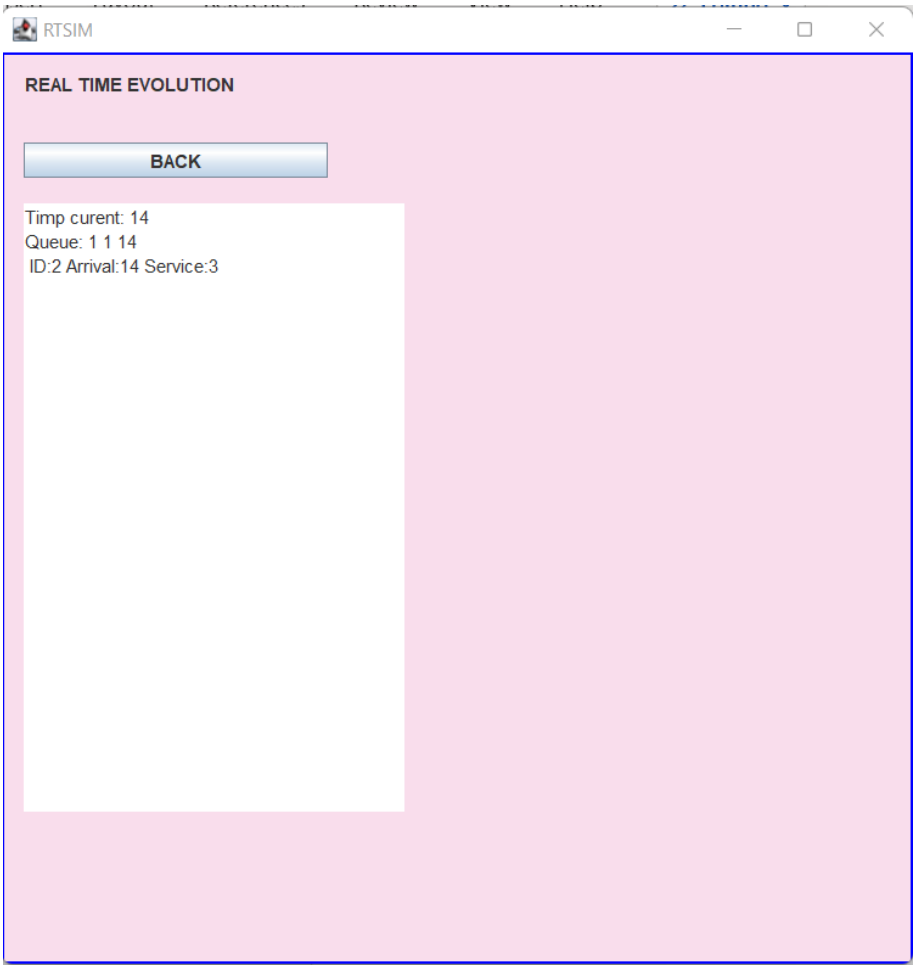
Metoda de run a threadului contine un iterator pentru timpul curent initializat cu 0. Se parcurge lista de clienti sortata in functie de timpul de ajungere, iar daca arrival time este egal cu timpul curent, se trimite clientul in coada in functie de strategia aleasa(se apeleaza metoda organizatorului) si este scos din lista de clienti generati. La fiecare ietrare pe lista se actualizeaza interfata de simulare in timp real cu clienti ramasi in asteptare, iar la fiecare adaugare de client in coada metoda actualizeaza si ea interfata cu persoanele ajunse in coada. Tot la fiecare iteratie si la fiecare adaugare de client se scrie intr-un fisier text exact aceelasi informatii din interfata grafica a simularii.

Interfetele grafice: UI si RTSim

Interfetele grafice au fost realizate prin intermediul bibliotecii JAVAXSwing dedicata manipularii GUI.



Interfata grafica de extragere a informatiilor este alcatuita din 8 text JTextFielduri si 8 JLabel-uri care deserversc la captarea informatiilor in legatura cu numarul de clienti, numarul de cozi disponibile, numarul maximi de clienti care pot sa stea la coada, minimul si maximul arrival time, minimul si maximul de service timp, un JComboBox pentru alegerea strategiilor si butonul de START care inchide fereastra si deschide interfata de afisare in timp real.



Interfata pentru afisarea datelor in timp real este formata dintr-un JLabel,un buton care inchide simularea si revine la interfata grafica initiala si o zona JTextArea care retine mai multe linii care este actualizata in timp real cu informatii despre clientii aflati in asteptare si in cozi. La fiecare a timpului curent din metoda de run a threadurilor zona se goleste si este actualizata cu informatii noi din lista de clienti generati sau clienti din coada.

5.Rezultate

Pentru testarea aplicatiei nu a fost folosit un framework de testare specific. Testarea acestei aplicatii poate sa fie incerta datorita generarii aleatorii a timpilor care urmeaza sa fie gestionati mai departe de metodele de adaugare a clienilor care gestioneaza timp minimi si dimensiuni de cozi(care la randulor ajung tot sa fie aleatoare).

Pentru obtinerea rezultatelor au fost intocmit un log-file pentru trei cazuri specifice

- Pentru 4 clienti, 2 cozi, timpul maxim de simulare de 60 de secunde, intervalul de ajungere de 2 - 30, intervalul de procesare a clientilor de 2 – 4
- Pentru 50 clienti, 5 cozi, timpul maxim de simulare de 60 de secunde, intervalul de ajungere de 2 - 40, intervalul de procesare a clientilor de 1 - 7
- Pentru 1000 clienti, 20 cozi, timpul maxim de simulare de 100 de secunde, intervalul de ajungere de 10 - 100, intervalul de procesare a clientilor de 3 – 9

Cele 3 fisiere au fost atasate proiectului.

6.Concluzii

In realizarea aplicatiei de gestionare a cozilor au fost folisite concepte noi de multithreading impreuna cu sincronizarea (folosirea structurilor de date safe pentru multithreading/sincronizare).

Threadurile sunt obiecte de lucru care executa aceelasi proces deodata pentru a veni in imbunatirea timpului de procesare pe CPU

Imbunatatiri

- Realizarea unei interfe grafice mai estetice
- Imbunatatirea corectitudinii algoritmului de adaugare a clientilor in coada in functie de activitatea threadului
- Dezvoltarea unui mediu de testare a aplicatiei
- Calcularea average time-ului si a momentului de peak a cozilor

7.Bibliografie

1.Siguranta threaduilor: https://www.baeldung.com/java-thread-safety

2. Gestionarea fisierelor text in Java : https://www.baeldung.com/java-write-to-file

3. Paradigme si design OOP: https://www.techtarget.com/searchapparchitecture/feature/An-intro-to-the-5-SOLID-principles-of-object-oriented-design

4. Diagrame UML si Threads – Curs OOP R. Brehar(anul 2 semestrul 1) - link preluat de pe moodle

5. Implementare – suport prezentare : https://dsrl.eu/courses/pt/materials/A2_Support_Presentation.pdf