

Simulare cozi

Documentație

Nume: Moș Robert

Grupa: 30228

Cuprins

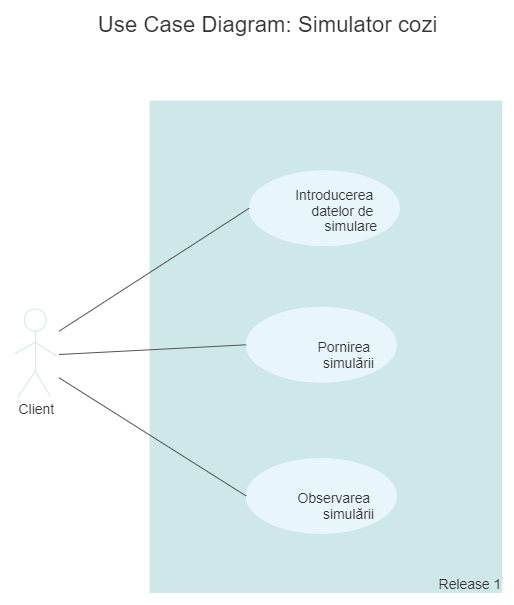
1. Obiectivul problemei
2. Analiza problemei
   1. Diagrama Use Cases
   2. Diagrama UML
3. Proiectarea problemei
4. Implementarea problemei
   1. Clase
   2. Metode
5. Testarea problemei
6. Concluzii
7. Bibliografie
8. Obiectivul problemei

Cerința acestei teme de laborator a fost implementarea în Java a unui program care simulează cozi de clienți și prin aplicarea mai multor metode în care clienții să fie așezați la coada să se reducă timpul de așteptare la coadă. Sunt simultați clienți care au câte un id, un timp de sosire și un timp de servire. In funcție de timpul la care ajung la coada, timpul de servire acești clienți sunt așezați la cozi. Problema are un scop foarte productiv pentru că la foarte multe magazine, aeroporturi, bănci timpul de asteptare la coadă este foarte mare și un astfel de program ar ajuta foarte mult la gestionarea cozilor la case, mai ales într-o pandemie cum este cea de față.

1. Analiza problemei

Codul scris pentru problemă a fost în mare parte construit pe baza prezentării de laborator, unde clienții erau asemănați cu task-urile, cozile erau asemănate cu serverele care primeau task-urile. Cozile primesc clienții de la un scheduler care verifică timpul de servire al clientului și îl asignează la o coadă pentru a fi servit. Scopul cozii este de a oferi unui client un loc pentru a fi servit la casa. Fiecărei cozi ii corespunde unul sau mai mulți clienți care așteaptă la o anumită coadă. Pentru implementarea codului trebuie să ne gândim după ce criterii sunt clienții asignați la coadă. Avem două criterii după care se vor putea organiza cozile pentru clienți, unul ar fi punerea clientului la coada cu cei mai puțini clienți care asteaptă să fie serviți. Cel de-al doilea criteriu după care pot fi puși aceștia la coadă este în funcție de timp, ei sunt puși la coada cu cel mai mic timp de așteptare.

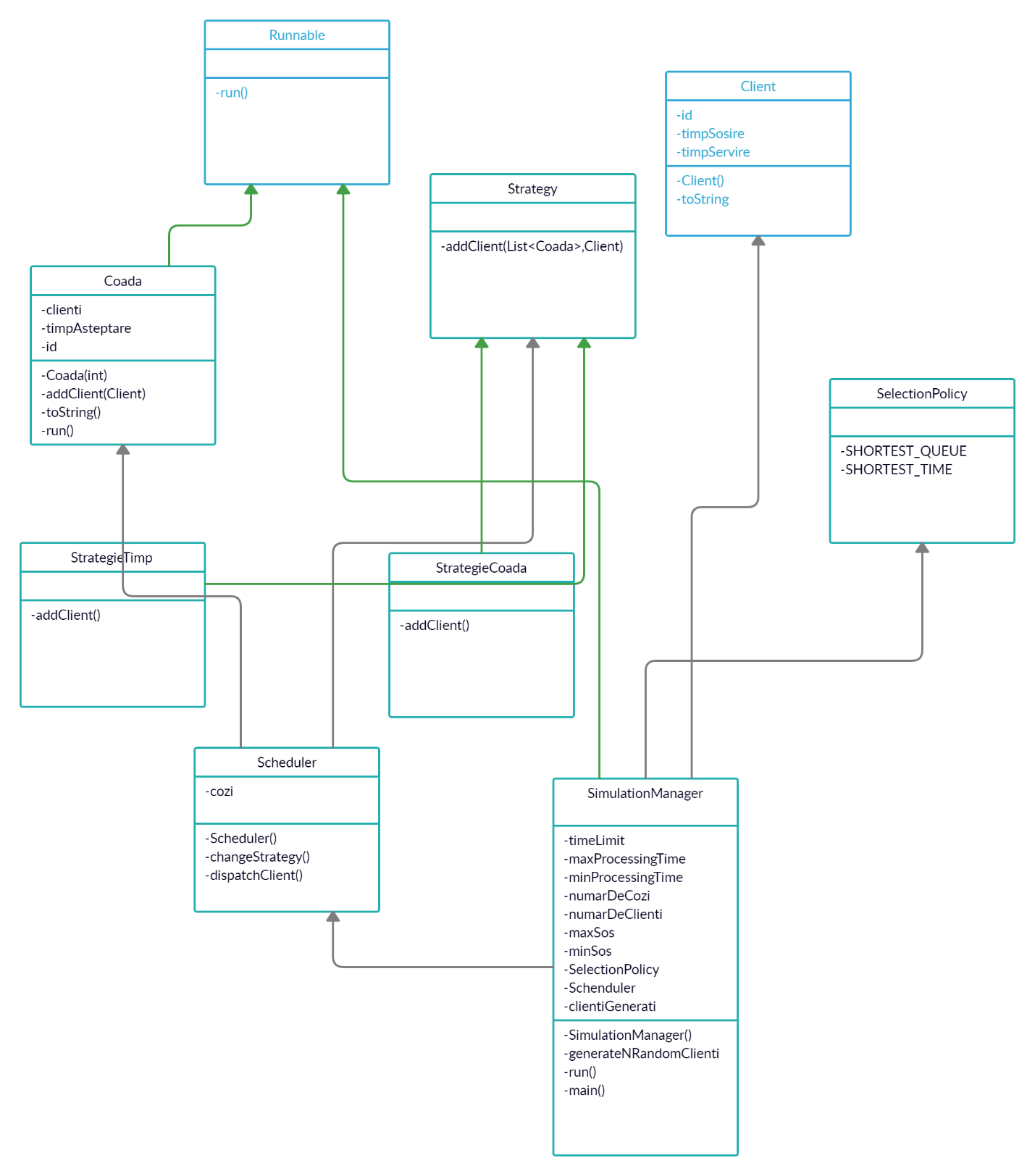
* 1. Diagrama Use Cases



În această diagramă este prezentat modul de adăugare a clientului la coadă în funcție de timpul de așteptare și numărul de clienți care așteptă la coada respectivă. Pentru fiecare client se va alege o strategie cea mai bună pentru servirea acestuia.

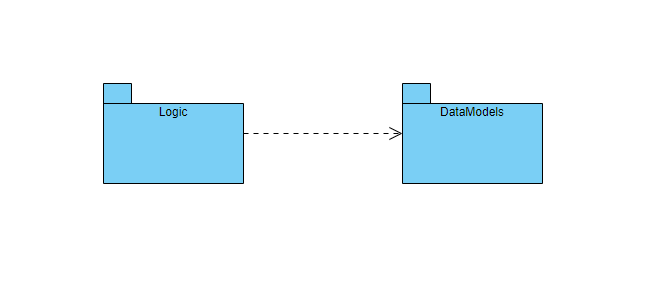
* 1. Diagrama de clase UML

Unified Modeling Language sau UML pe scurt este un limbaj standard pentru descrierea de modele si specificatii pentru software. UML a fost la bază dezvoltat pentru reprezentarea complexității programelor orientate pe obiect, al căror fundament este structurarea programelor pe clase, și instanțele acestora ( numite și obiecte ). Diagrama UML ajută foarte mult la implementarea unui program pentru că iti poți face o idee mai clară la cum va arăta programul tău si va fi mai ușor de implementat.



1. Proiectarea problemei

Pachetele Java ajută la organizarea mai multor module și grupează clase și interfețe. Eu pentru această problemă am folosit doar doua pachete, unul numit Logic in care am introdus clasele SelectionPolicy, SimulationManager, StrategieCoada, StrategieTimp, Strategy iar unul numit DataModels în care am introdus clasele Client, Coada, Scheduler. Organizarea claselor în package-uri este foarte utilă pentru că oferă un aspect mai ordonat și ajută să înțelegem mai bine despre ce este vorba în programul respectiv.



1. Implementarea problemei
   1. Clase

Pentru rezolvarea problemei propuse a fost nevoie de folosirea mai multor clase, construcția claselor a fost făcută după cea din prezentarea de laborator.

Clasa Client: Această clasă conține toate informațiile despre un client, cum ar fi id, timp de servire și timp de sosire. Id-ul este un număr unic al fiecărui client de la 1 la n(numărul de clienți). Timpul de servire este timpul la care clientul va intra în coadă, iar timpul de sosire este timpul pe care trebuie să îl aștepte clientul pentru a fi servit.

Clasa Coada: În această clasă se găsesc metode pentru adăugarea clienților în coadă.

Clasa Scheduler: În această clasă se va organiza punerea în coadă a clienților după criteriile de timp mai mic sau coadă mai scurtă.

Clasa SimulationManager: În această clasă în care se află și metoda main se simulează aplicația. Aici avem si datele de intrare timelimit,este timpul de câte ori se va simula programul, maxProcessingTime este timpul maxim de procesare pentru client, minProcessingTime este timpul minim de procesare pentru client, numarDeCozi este numărul care ne arată câte cozi se vor creea, numarDeClienti ne arată câți clienți vor aștepta să fie băgați în cozi, maxSos care reprezintă timpul maxim de sosire, minSos care este timpul minim de sosire.

Clasa StrategieCoada: Este clasa în care atribuim clientul unei cozi în funcție de lungimea acesteia.

Clasa StrategieTimp: Este clasa în care punem un client în coadă în funcție de cel mai mic timp de așteptare.

* 1. Metode

Metode din clasa Client:

public Client(int id, int timpSosire, int timpServire)  
{  
 this.id=id;  
 this.timpSosire=timpSosire;  
 this.timpServire=timpServire;  
}

În această metodă se atribuie id-ului, timpului de sosire si timpului de servire câte o valoare.

public String toString()  
{  
 return "("+id+","+timpSosire+","+timpServire+")";  
}

Această metodă ajută la fișarea id-ului, timpului de sosire si timpului de servire a clienților

Metode din clasa Coada:

public void addClient(Client newClient) {  
 clienti.add(newClient);  
  
 timpAsteptare.addAndGet(newClient.getTimpServire());  
  
}

Prin această metodă se adaugă la lista de clienți timpul de asteptare respectiv unui client și clientul.

public String toString()  
{  
 String s="";  
 s=s+"Coada "+id;  
 for (Client c: clienti) {  
 s=s+c.toString()+"; ";  
 }  
  
 return s;  
}

S-a folosit această metodă pentru afișarea cozii și clienților care aparțin de coada respectivă intr-un mod mai organizat astfel încât să se vadă mult mai clar ce va afișa programul.

public void run() {  
 while (true) {  
 if (clienti.size() > 0) {  
 Client newClient = clienti.peek();  
 int timp=newClient.getTimpServire();  
 for (int i = 1; i <= timp; i++) {  
 try {  
 Thread.*sleep*(1000);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 timpAsteptare.decrementAndGet();  
 newClient.setTimpServire(newClient.getTimpServire()-1);  
  
 }  
 try {  
 clienti.take();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
}

În această metodă se verifică dacă numărul de clienți nu este null si apoi se asteaptă câte o secundă până la terminarea timpului de servire, după care timpul de asteptare va scădea, la fel si timpul de servire. La final se va șterge clientul.

Metode din clasa Scheduler:

public Scheduler(int maxCozi, int maxClientiPeCoada)  
{  
 cozi=new ArrayList<Coada>();  
 for(int i=0;i<maxCozi;i++)  
 {  
 Coada c=new Coada(i+1);  
 cozi.add(c);  
 Thread t=new Thread(c);  
 t.start();  
 }  
}

În această metodă se initializeaza o coadă, apoi se adaugă o coadă in lista de cozi si se începe thredul.

public void changeStrategy(SelectionPolicy policy)  
{  
 if(policy==SelectionPolicy.*SHORTEST\_QUEUE*)  
 {  
 strategy =new StrategieCoada();  
 }  
 if(policy==SelectionPolicy.*SHORTEST\_TIME*){  
 strategy =new StrategieTimp();  
 }  
}

În această metodă se alege criteriul de adăugare a clientului în coadă în funcție de coada cea mai scurtă sau timpul de asteptare cel mai mic.

Metode din clasa SimulationManager:

public SimulationManager()  
{  
 Scheduler sc=new Scheduler(numarDeCozi, numarDeClienti);  
 sc.changeStrategy(selectionPolicy);  
 generateNRandomClienti();  
 this.scheduler=sc;  
  
}

În această metodă se inițializează scheduler-ul si se generează clienții random.

private void generateNRandomClienti()  
{  
 Random rand= new Random();  
 for(int i=0;i<numarDeClienti;i++)  
 {  
 int timpSos=(int) (Math.*random*()\*(maxSos-minSos+1)+minSos);  
 int procT=(int) (Math.*random*()\*(maxProcessingTime-minProcessingTime+1)+minProcessingTime);  
 Client c=new Client(i+1,timpSos,procT);  
 clientiGenerati.add(c);  
 }  
 Collections.*sort*(clientiGenerati);  
}

Această metodă generează un număr de clienți random și pentru fiecare client se generează câte un timp de sosire random si un timp de procesare random, apoi sunt adaugati in listă si sortați.

public void run()  
{  
 int currentTime=0;  
 while(currentTime<timeLimit)  
 {  
 System.*out*.println("-----Time: "+currentTime+"-----");  
 ArrayList<Client> clientiStersi=new ArrayList<Client>();  
 for (Client c: clientiGenerati)  
 {  
 if(c.getTimpSosire()==currentTime)  
 {  
 scheduler.dispatchClient(c);  
 clientiStersi.add(c);  
 }  
 }  
 for (Client c: clientiStersi){  
 clientiGenerati.remove(c);  
 }  
 System.*out*.print("Clienti: ");  
 for (Client c: clientiGenerati)  
 {  
 System.*out*.print(c.toString());  
 }  
 System.*out*.println();  
 for (Coada c: scheduler.getCozi())  
 {  
 System.*out*.println(c.toString());  
 }  
 currentTime++;  
 try {  
 Thread.*sleep*(1000);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
  
}

În metoda run se verifică dacă timpul curent e mai mic decat timpul limită, apoi se afisează timpul curent. Se parcurg clienții generați și se verifiă dacă timpul de sosire al clientului este egal cu timpul curent, se va adăuga în coadă iar clientul va fi adăugat într-o altă listă, apoi se va parcurge lista cu clienții care trebuie ștersi si va fi sters, după care se vor afișa clienții rămași si cozile. Se va crește timpul curent si se va astepta o secundă.

Metode din clasa StrategieCoadă:

public void addClient(List<Coada> cozi, Client c) {  
 Coada cod = cozi.get(0);  
 for (Coada cod2 : cozi) {  
 if (cod2.getClient().size() < cod.getClient().size())  
 cod = cod2;  
 }  
 cod.addClient(c);  
}

Prin această metodă se va alege coada cu cei mai puțini clienți și se va adăuga clientul la coada cea mai scurtă.

Metode din clasa StrategieTimp:

public void addClient(List<Coada> cozi, Client c) {  
 Coada cod = cozi.get(0);  
 for (Coada cod2 : cozi) {  
 if (cod2.getTimpAsteptare() < cod.getTimpAsteptare())  
 cod = cod2;  
 }  
 cod.addClient(c);  
}

În această metodă se va adăuga clientul in coada cu timpul de așteptare cel mai mic.

1. Rezultatele problemei

În ceea ce privește testarea problemei, programul meu funcționează fără nicio problemă. Pentru testarea problemei este nevoie să se dea o valoarea pentru numărul de cozi care să se genereze, o valoare pentru numărul de clienți care să fie generați, o valoare pentru timpul care ne spune de cate ori să rulăm programul, o valoare pentru timpul minim si maxim de procesări și o valoare pentru timpul minim si maxim de sosiri. Programul va genera random un număr de clienți care va fi egal cu cel introdus, fiecare client va avea câte un id unic, un timp de servire random cuprins între valorile date în timpul minim si maxim de procesare si un timp de sosire generat random pentru valorile cuprinse între timul de sosire minim si maxim.

Pentru exemplul din laborator sunt prezentate câteva exemple de ieșiri.

-----Time: 0-----

Clienti: (4,3,4)(3,5,7)(1,14,8)(2,28,17)

Coada 1

Coada 2

-----Time: 1-----

Clienti: (4,3,4)(3,5,7)(1,14,8)(2,28,17)

Coada 1

Coada 2

-----Time: 2-----

Clienti: (4,3,4)(3,5,7)(1,14,8)(2,28,17)

Coada 1

Coada 2

-----Time: 3-----

Clienti: (3,5,7)(1,14,8)(2,28,17)

Coada 1(4,3,4);

Coada 2

-----Time: 4-----

Clienti: (3,5,7)(1,14,8)(2,28,17)

Coada 1(4,3,3);

Coada 2

-----Time: 5-----

Clienti: (1,14,8)(2,28,17)

Coada 1(4,3,2);

Coada 2(3,5,7);

-----Time: 6-----

Clienti: (1,14,8)(2,28,17)

Coada 1(4,3,1);

Coada 2(3,5,6);

-----Time: 7-----

Clienti: (1,14,8)(2,28,17)

Coada 1

Coada 2(3,5,5);

-----Time: 8-----

Clienti: (1,14,8)(2,28,17)

Coada 1

Coada 2(3,5,4);

-----Time: 9-----

Clienti: (1,14,8)(2,28,17)

Coada 1

Coada 2(3,5,3);

1. Concluzii

Având în vedere tema propusă, a fost una foarte interesantă care mi-a dat destul de mult de gandit datorită introducerii unor noi concepte. De asemenea a fost o temă foarte practică având legătură cu ce se întâmplă în viața reală. O astfel de aplicație ar gestiona mult mai bine aglomerațiile din super-market-uri mai ales în perioada aceasta de pandemie, ar reduce mult mai bine cazurile infectărilor cu noul virus. Desi pentru implementarea unei asemenea aplicații care să funcționeze foarte bine în lumea reală trebuie luați foarte mulți factori in seamă. Datorită lipsei de timp nu am reușit să implementez și o interfață grafică, sunt sigur ca dacă reușeam iesea un program cu mult mai reușit.

1. Bibliografie

Ca surse de informare am folosit: Youtube, Wikipedia

Qt qt qt qt qt qt qt qt qt qt qt qt qt q t qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt q t qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt q t qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt q t qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt q t qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt q t qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt q t qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt q t qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt q t qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt q t qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt q t qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt q t qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt qt q t qt qt