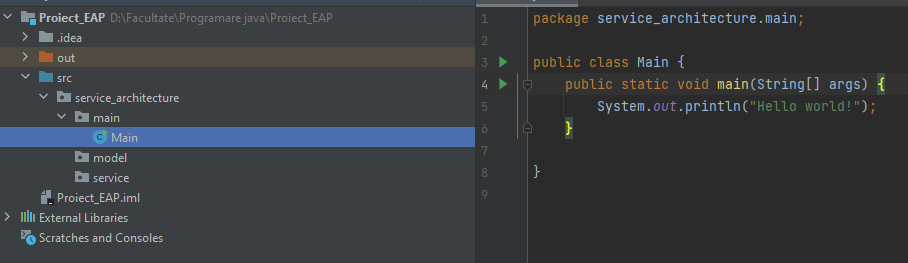
# PAO Project – E-ticketing platform

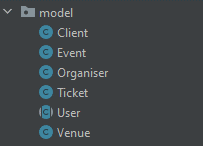
Popescu Mihnea – 264

# Initierea obiectelor

Pentru inceput, am creat package-ul service\_architecture

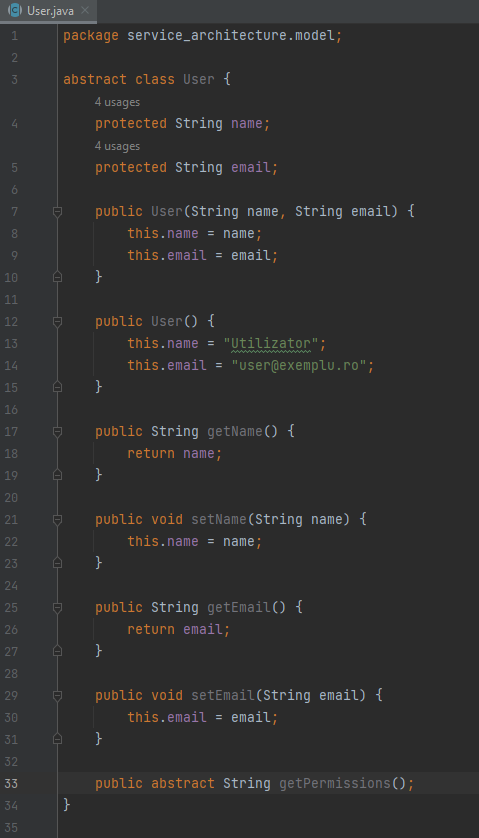


Am adaugat si subpackage-urile main (ce contine clasa principala in care se apeleaza metodele serviciilor), model (in care vom adauga clasele obiectelor) si service (in care avem operatiile de serviciu).

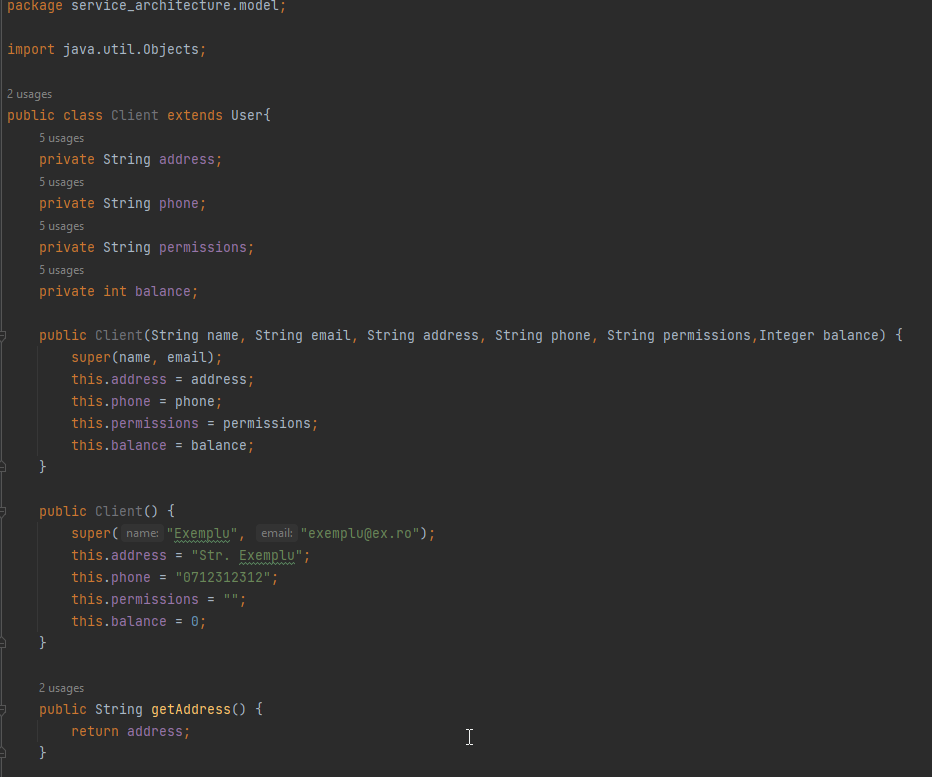


Modelele folosite in implementarea platformei de e-ticketing sunt urmatoarele:

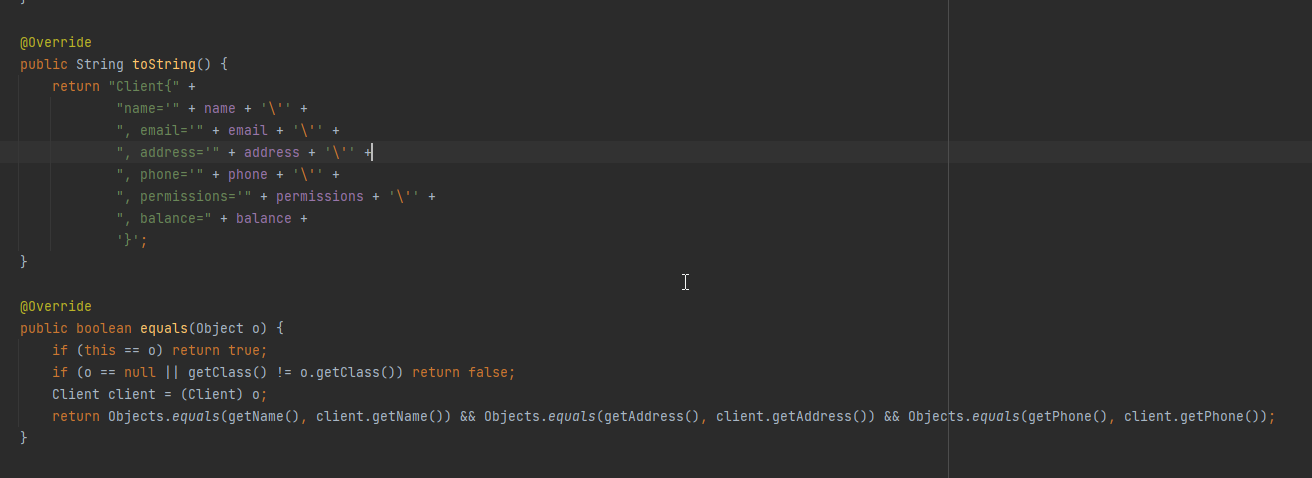
* User (clasa de baza a unui utilizator)
* Client (mosteneste clasa User si reprezinta un utilizator care poate cumpara bilete)
* Organiser (mosteneste clasa User si reprezinta un utilizator care poate organiza evenimente)
* Venue (reprezinta o locatie unde poate avea loc un eveniment si contine detalii precum nr de locuri)
* Event (reprezinta un eveniment, acestuia trebuie sa-i fie atribuit un Organiser si un Venue)
* Ticket (reprezinta un bilet de acces la un eveniment)



Am inceput prin crearea clasei abstracte User, ce defineste un utilizator de baza al platformei (un cont). Aceasta clasa contine si metoda abstracta getPermissions, ce va fi folosite de clasele ce o vor mosteni.



Am adaugat clasei Client atributele address, phone, permissions si balance.



Am facut suprascriere la metodele toString si equals pentru a include atributele speciale. 2 instante ale clasei Client pot fi egale daca au acelasi nume, aceeasi adresa si acelasi numar de telefon.

Asemanator am creat si celelalte clase: Organiser, Ticket, Event si Venue.

Am adaugat in clasa User un ArrayList de tipul <Ticket>

protected ArrayList<Ticket> tickets;

# Citirea si stocarea datelor

In package-ul service\_architecture.service am creat un nou package: fileio.

In el am definit clasa singleton VenueReader unde am efectuat citirea din fisierul storage/venue\_list.csv

public class EventReader {  
 private static EventReader *instance*;  
 private EventReader(){}  
  
 public static EventReader getInstance() {  
 if(*instance* == null) {  
 *instance* = new EventReader();  
 }  
 return *instance*;  
 }  
  
 private static String *line* = "";  
  
 public static ArrayList<Event> readFile(String basePath) {  
 ArrayList<Event> events = new ArrayList<Event>();  
 try(BufferedReader br = new BufferedReader(new FileReader(basePath+"event\_list.csv"))) {  
 while((*line* = br.readLine()) != null) {  
 String[] values = *line*.split(",");  
 if(values.length > 0) {  
 Event event = new Event();  
 event.setOrganiser(Integer.*parseInt*(values[0]));  
 event.setName(values[1]);  
 event.setDate(values[2]);  
 event.setDescription(values[3]);  
 event.setTicket\_price(Integer.*parseInt*(values[4]));  
 event.setVenue(Integer.*parseInt*(values[5]));  
 events.add(event);  
 }  
 }  
 } catch(IOException e) {  
 e.printStackTrace();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 return events;  
 }  
}

Asemanator am facut operatiile de citire si pentru organisers, events and clients si tickets.

Am creat o noua clasa singleton in package-ul service\_architecture.service.fileio numita GetCSVData in care am adaugat metoda getCSVInputData ce returneaza un Map ce contine ArrayList-uri cu toate obiectele salvate in fisierele CSV.

public static Map<String, ArrayList> getCSVInputData(String basePath) {  
 Map map = new HashMap<String, ArrayList>();  
  
 VenueReader vr = VenueReader.*getInstance*();  
 map.put("venues",vr.*readFile*(basePath));  
  
 OrganiserReader or = OrganiserReader.*getInstance*();  
 map.put("organisers", or.*readFile*(basePath));  
  
 EventReader er = EventReader.*getInstance*();  
 map.put("events", er.*readFile*(basePath));  
  
 ClientReader cr = ClientReader.*getInstance*();  
 map.put("clients", cr.*readFile*(basePath));  
  
 TicketReader tr = TicketReader.*getInstance*();  
 map.put("tickets", tr.*readFile*(basePath));  
  
 return map;  
}

Iar in clasa service\_packages.main.Main am adaugat atributele clients, organisers, venues si events, ce reprezinta liste unice ale instantelor acestor obiecte ce vor putea fi prelucrate de aplicatie.

private String basePath = "D:\\facultate\\an2\\EAP\\Proiect\_EAP\\src\\storage\\";  
  
private ArrayList<Client> clients;  
private ArrayList<Organiser> organisers;  
private ArrayList<Venue> venues;  
  
private ArrayList<Event> events;  
  
private void getData() {  
 String basePath = this.basePath;  
  
 //inputs  
 GetCSVData singleton = GetCSVData.*getInstance*();  
  
 Map<String, ArrayList> map = singleton.*getCSVInputData*(basePath);  
  
 if(map.containsKey("venues")) {  
 this.setVenues(map.get("venues"));  
 }  
 System.*out*.println("Found " + this.getVenues().size() +" venues.");  
  
 if(map.containsKey("organisers")) {  
 this.setOrganisers(map.get("organisers"));  
 }  
 System.*out*.println("Found " + this.getOrganisers().size() + " organisers.");  
  
 if(map.containsKey("events")) {  
 this.setEvents(map.get("events"));  
 }  
 System.*out*.println("Found "+ this.getEvents().size() + " events.");  
  
 if(map.containsKey("clients")) {  
 this.setClients(map.get("clients"));  
 }  
 System.*out*.println("Found " + this.getClients().size() + " clients.");  
  
 if(map.containsKey("tickets")) {  
 ArrayList<Ticket> tickets = map.get("tickets");  
  
 for(Client client : this.getClients()) {  
 ArrayList<Ticket> clientTickets = new ArrayList<Ticket>();  
 for(Ticket ticket : tickets) {  
 if(ticket.getUser() == client.hashCode()) {  
 clientTickets.add(ticket);  
 }  
 }  
 client.setTickets(clientTickets);  
 }  
  
 for(Organiser organiser : this.getOrganisers()) {  
 ArrayList<Ticket> organiserTickets = new ArrayList<Ticket>();  
 for(Ticket ticket : tickets) {  
 if(ticket.getUser() == organiser.hashCode()) {  
 organiserTickets.add(ticket);  
 }  
 }  
 organiser.setTickets(organiserTickets);  
 }  
  
 System.*out*.println("Found " + tickets.size() + " tickets.");  
 }  
}

In cazul obiectelor Tickets, le-am prelucrat pe fiecare in parte si le-am adaugat in ArrayList-urile User-ilor respective.

Pentru scrierea datelor in fisierl, am creat in package-ul service\_architecture.service.fileio clasa singleton WriteCSVData:

private static WriteCSVData *instance*;  
private WriteCSVData(){}  
  
public static WriteCSVData getInstance() {  
 if(*instance* == null) {  
 *instance* = new WriteCSVData();  
 }  
 return *instance*;  
}

Am creat metoda pentru salvarea instantelor obiectelor de tip Venue:

public static void writeVenues(String basePath, ArrayList<Venue> venues) {  
 File f = new File(basePath, "venue\_list.csv");  
 try{  
 if(f.createNewFile()) {  
 System.*out*.println("venue\_list.csv file created");  
 }  
 } catch (IOException e) {  
 System.*out*.println(e);  
 }  
  
 try(BufferedWriter bw = new BufferedWriter(new FileWriter(basePath+"venue\_list.csv"))) {  
 String line;  
 for(Venue venue : venues) {  
 line = venue.getName() + "," + venue.getAddress() + "," + venue.getCapacity().toString() + "," + venue.getOpened().toString();  
 bw.newLine();  
 bw.write(line);  
 }  
 } catch(IOException e) {  
 e.printStackTrace();  
 }  
 System.*out*.println("Saved " + venues.size() + " venues.");  
}

Asemanator am facut si pentru events, clients si organisers.

Pentru Tickets, am creat 2 metode separate pentru a salva array-ul tickets din Clasele Client si Organiser.

public static void writeEvents(String basePath, ArrayList<Event> events) {  
 File f = new File(basePath, "event\_list.csv");  
 try{  
 if(f.createNewFile()) {  
 System.*out*.println("event\_list.csv file created");  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 try(BufferedWriter bw = new BufferedWriter(new FileWriter(basePath+"event\_list.csv"))) {  
 String line;  
 for(Event event : events) {  
 line = event.getOrganiser().toString() + "," + event.getName() + "," + event.getDate() + "," + event.getDescription() + "," + event.getTicket\_price().toString() + "," + event.getVenue();  
 bw.write(line);  
 bw.newLine();  
 }  
 } catch(IOException e) {  
 e.printStackTrace();  
 }  
 System.*out*.println("Saved " + events.size() + " events.");  
}

public static void writeTicketsOrganiser(String basepath, Organiser organiser) {  
 ArrayList<Ticket> tickets = organiser.getTickets();  
 try(BufferedWriter bw = new BufferedWriter(new FileWriter(basepath+"ticket\_list.csv", true))) {  
 String line;  
 for(Ticket ticket : tickets) {  
 line = organiser.hashCode() + "," + ticket.getEvent() + "," + ticket.getPrice().toString() + "," + ticket.getDate\_bought() + "," + ticket.getTicket\_type().toString() + "," + ticket.getUsed().toString() + "," + ticket.getId();  
 bw.write(line);  
 bw.newLine();  
 *ticketCount*++;  
 }  
 }  
 catch (IOException e) {  
 e.printStackTrace();  
 }  
}

Apoi am creat o metoda statica pentru a combina toate celelalte metode si a le apela pe toate deodata:

public static void saveData(String basePath, ArrayList<Venue> venues, ArrayList<Organiser> organisers, ArrayList<Event> events, ArrayList<Client> clients) {  
 System.*out*.println("SAVING DATA.");  
 *writeVenues*(basePath, venues);  
  
 // Reset the ticket\_list.csv file  
 File ticketList = new File(basePath, "ticket\_list.csv");  
 try {  
 if(ticketList.createNewFile()) {  
 System.*out*.println("ticket\_list.csv file created");  
 }  
 else {  
 // Delete all contents from the ticket\_list.csv  
 new FileWriter(basePath+"ticket\_list.csv", false).close();  
 }  
 }  
 catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 *ticketCount* = 0;  
  
 *writeOrganisers*(basePath, organisers);  
 *writeEvents*(basePath, events);  
 *writeClients*(basePath, clients);  
  
 System.*out*.println("Saved " + *ticketCount* + " tickets.");  
}

# Meniul principal

In interiorul clasei Main din service\_architecture.main am creat metoda menu, unde vom afisa meniul sistemului de E-ticketing:

1. Logarea ca client – selecteaza un client existent sau inregistreaza unul nou si acceseaza operatiile permise acestuia.
2. Logarea ca organizator
3. Operatii de administrare a sistemului
4. Inchiderea programului – opreste executia programului fara a salva datele.

System.*out*.println("\*");  
System.*out*.println("\*");  
System.*out*.println("\*");  
System.*out*.println("E-TICKETING SYSTEM");  
System.*out*.println("1. LOGIN AS CLIENT");  
System.*out*.println("2. LOGIN AS ORGANIZER");  
System.*out*.println("3. SYSTEM ADMINISTRATION");  
System.*out*.println("0. EXIT");  
System.*out*.println("Enter your input...");

# Meniul administrativ

Meniul administrativ are urmatoarele optiuni:

1. Salvarea datelor
2. Administrarea locatiilor

Salvarea datelor inseamna salvarea modificarilor din memoria aplicatiei in fisierele .csv pentru a fi stocate.

Administrarea locatiilor afiseaza un meniu cu urmatoarele optiuni:

* Selecteaza o locatie: selectarea unei locatii va deschide meniul editarii locatiei. Toti parametrii obiectului Venue pot fi editati in meniul respectiv, iar la finalizarea procesului de editare, acesta va fi salvat in memoria aplicatiei, iar evenimentele si tichetele asociate acelui eveniment vor fi actualizate cu hashCode-ul nou. (efect de cascada)
* Adauga o locatie: aceasta optiune deschide formularul de creare a unei locatii. Dupa finalizarea acestei operatii, locatia va fi adaugata automat in lista din memoria aplicatiei.

# Meniul clientilor

Meniul clientilor are urmatoarele optiuni:

* Selecteaza un client existent: afiseaza meniul actiunilor pe care le poate efectua un client
* Inregistreaza un client: creeaza un client nou si il introduce in memoria aplicatiei
* Intoarcere la meniul anterior

Selectarea optiunii de inregistrare a unui client va deschide formularul prin care se pot completa toate detaliile necesare contului unui client (nume, adresa e-mail, adresa de domiciliu, numar de telefon). Formularul se afla in clasa service\_architecture.service.CreateClient.java

Selectarea unui client existent va deschide meniul de control al clientului, ce are urmatoarele actiuni posibile:

* Achizitionarea unui bilet: afiseaza lista cu evenimentele disponibile, iar daca utilizatorul va alege un eveniment, va apela clasa BuyTicket si metoda ProcessTransaction din aceasta. In metoda ProcessTransaction este folosita o exceptie custom numita ClientNotEnoughBalanceException, apelata in cazul in care clientul nu are destui bani in balanta. Pretul biletului va fi scazut din balanta clientului si adaugat in balanta organizatorului.

try {  
 this.validateTransaction();  
}  
catch (ClientNotEnoughBalanceException e) {  
 System.*out*.println("Error: " + e);  
 return false;  
}

* Vizualizare bilete: afiseaza lista biletelor achizitionate de catre utilizator, iar selectarea tichetului va afisa ID-ul unic de identificare al biletului pentru a putea fi verificat de organizator.
* Modificarea detaliilor: deschide formularul de editare al clientului, unde acesta va alege atributul pe care vrea sa il editeze si il va putea modifica.
* Depunerea de bani: apeleaza metoda statica processTransaction din clasa DepositMoney, aceasta primeste suma de bani pe care doreste clientul sa o adauge in cont, iar daca aceasta este negativa, va trimite o exceptie de tipul NotMoneyNumber:

try {  
 *validateMoney*(amount);  
} catch (NotMoneyNumber e) {  
 System.*out*.println("Error: " + e);  
 return;  
}

Daca suma de bani este valida, atunci clientului o sa ii fie cerute informatiile de la cardul de credit/debit pentru a putea procesa tranzactia. Dupa asta, clientului o sa ii fie adaugati in balanta suma de bani.

# Meniul organizatorilor

Meniul organizatorilor are urmatoarele optiuni:

* Selecteaza un organizator: afiseaza meniul actiunilor pe care il poate face respectivul organizator
* Inregistreaza un nou organizator: afiseaza formularul de creare al unui nou organizator, apoi il introduce in lista organizatorilor din memoria aplicatiei. Formularul este realizat prin intermediul interfetei CreateO din service\_architecture.service\_interfaces implementata in CreateOrganiser.
* Intoarcere la meniul anterior

Dupa ce selectam un organizator, pe ecran este afisat numele organizatorului si balanta de bani pe care o are in cont si avem urmatoarele optiuni:

* Administrare evenimente
* Creeaza eveniment
* Editeaza contul de organizator
* Intoarcere la meniul anterior

Optiunea de editare a contului de organizator apeleaza metoda edit din EditOrganiser, ce implementeaza interfata EditO. In aceasta metoda, sunt afisate atributele ce pot fi editate de catre organizator. Pentru a edita un atribut, scriem numarul de criteriu asociat acestuia si introducem valoarea noua dorita. La final, scriem 0 pentru a ne intoarce in ecranul anterior si modificarile vor fi salvate automat. Dupa ce salvam modificarile, toate evenimentele si biletele vor fi actualizate cu noile hashcode-uri.

Crearea unui eveniment instantiaza un obiect al clasei CreateEvent si apeleaza apoi metoda create din acesta. Procesul de creare al unui eveniment este asemanator cu cele precedente, singura diferenta este faptul ca la inceput organizatorului ii sunt aratate toate locatiile ce sunt disponibile.

# Administrarea evenimentelor

In meniul de procesare a unui eveniment, avem urmatoarele optiuni:

* Procesare tichete
* Ofera tichete

Optiunea de procesare tichete este folosita de un organizator de evenimente pentru a verifica biletele folosite de participantii la eveniment. Aceasta verifica daca exista un bilet cu ID-ul respectiv, iar daca acesta exista, apartine acestui eveniment si nu a mai fost folosit, atunci il marcheaza ca fiind folosit.

Optiunea de oferire a tichetelor functioneaza apeleaza metoda giveawayTickets si primeste de la organizator adresa de e-mail a utilizatorului care va primi biletul. In cazul in care utilizatorul exista, acesta va primi un bilet pentru eveniment. Daca nu exista, atunci pe ecran se va afisa o eroare.

# Auditing

Pentru realizarea serviciului de auditing, am creat clasa de tip singleton AuditLog in package-ul service\_architecture.service.audit.

In clasa AuditLog am declarat un ArrayList cu elemente de tip Map, unde vom inregistra log-urile.

De fiecare data cand o actiune este efectuata, aceasta va fi salvata in ArrayList-ul logs din AuditLog prin apelarea metodei log.

Atunci cand modificarile din aplicatie vor fi salvate in fisier, si log-urile din AuditLog vor fi salvate in fisierul storage/audit\_log.csv prin intermediul metodei writeData.

public void log(String text) {  
 Map element = new HashMap<String, String>();  
 element.put("text", text.replace(",",""));  
 element.put("date", new SimpleDateFormat("dd/MM/yyyy HH:mm:ss").format(new Date()));  
 this.getLogs().add(element);  
}

public void writeData(String basePath) {  
 File f = new File(basePath, "audit\_log.csv");  
 try{  
 if(f.createNewFile()) {  
 System.*out*.println("audit\_log.csv file created");  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 try(BufferedWriter bw = new BufferedWriter(new FileWriter(basePath+"audit\_log.csv",true))) {  
 String line;  
 for(Map m : this.getLogs()) {  
 if(m.containsKey("text") && m.containsKey("date")) {  
 line = m.get("text") + "," + m.get("date");  
 bw.write(line);  
 bw.newLine();  
 }  
 }  
 } catch(IOException e) {  
 e.printStackTrace();  
 }  
 System.*out*.println("Saved " + this.logs.size() + " audit logs.");  
 this.setLogs(new ArrayList<Map>());  
}

# Final

Am adaugat clasa Main.java in folderul sursa pentru a apela direct metoda main din Clasa service\_architecture.main.