Java Advanced – Herhalingsoefening: Ticket Systeem

Opdracht

In deze opdracht maak je een **ticket systeem** voor concertzalen. Er kunnen optredens (*events*) aangemaakt worden en aan elk optreden kan de locatie toegevoegd worden waar dat optreden zal doorgaan. Gebruikers kunnen tickets bestellen voor optredens en komen na het bestellen in een wachtrij terecht, die dan door de organisatoren gebruikt kan worden om tickets toe te wijzen.

Clone het startproject op https://github.com/PXLJavaAdvanced1819/week9-herhaling

We hebben al een aantal basisklassen voorzien voor jullie, zodat jullie hiermee geen tijd verliezen.
De **User** klasse stelt een gebruiker in het ticket systeem voor, met bijhorende member variabelen.
Een **Venue** stelt een concertzaal of -locatie voor. Onder andere locatie en capaciteit kunnen hierin worden bijgehouden.
Een **Event** wordt gebruikt om een optreden van een muziekband, theatergroep, … op te slaan, met gegevens als naam, beschrijving, tijdstip, locatie (*Venue*) en een lijst van *attendees*. (*User*s die aanwezig zullen zijn op het optreden)
Elke instantie van bovenstaande klassen heeft ook een **uniek ID**, waarmee zo'n instantie geïdentificeerd kan worden.

QueueService

Maak een **QueueService** klasse. Deze klasse moet een **lijst van wachtrijen** bijhouden, namelijk één voor elk event. Gebruik hiervoor een **combinatie** van collections, die toelaat om de gegevens op deze manier op te slaan. De wachtrijen zullen nadien gebruikt worden om te bepalen welke gebruikers eerst een ticket toegewezen krijgen voor een bepaald event. (*first come, first served*)
Voorzie minstens volgende methodes:

- *addToQueue(eventID, user)*
  - om een gebruiker aan de wachtrij van het gegeven event toe te voegen
  - indien er nog geen wachtrij bestaat voor dit event, wordt deze aangemaakt
- *getQueue(eventId)*
  - geeft de wachtrij voor het gegeven event terug
- *getNextInLine(eventID)*
  - geeft de eerstvolgende gebruiker in de wachtrij terug voor het gegeven event
  - de gebruiker wordt **niet** verwijderd uit de wachtrij
- *removeFromQueue(eventID)*
  - verwijdert de gebruiker die eerst in de wachtrij stond voor het gegeven event
- *printQueue(eventID)*
  - print de wachtrij van het gegeven event
- *getQueueSize(eventID)*
  - geeft het aantal personen in de wachtrij voor het gegeven event terug

TicketSystem

Schrijf nu de klasse **TicketSystem**. In de constructor van deze klasse wordt een instantie van *QueueService* aangemaakt, die als member variabele wordt opgeslagen.
Verder houdt deze klasse een lijst van users bij, een lijst van locaties en een lijst van events. Voorzie voor elk van deze lijsten de nodige methoden om items er aan toe te voegen. (*addUser, addEvent, addVenue*)
Kies een gepaste collection om deze lijsten bij te houden, zodat je *Users, Events en Venues* kan opvragen aan hand van hun unieke ID. Schrijf hier ook de methoden voor: *getUser(id), getEvent(id)* en *getVenue(id)*.

Verder moeten volgende methoden in deze klasse zitten:

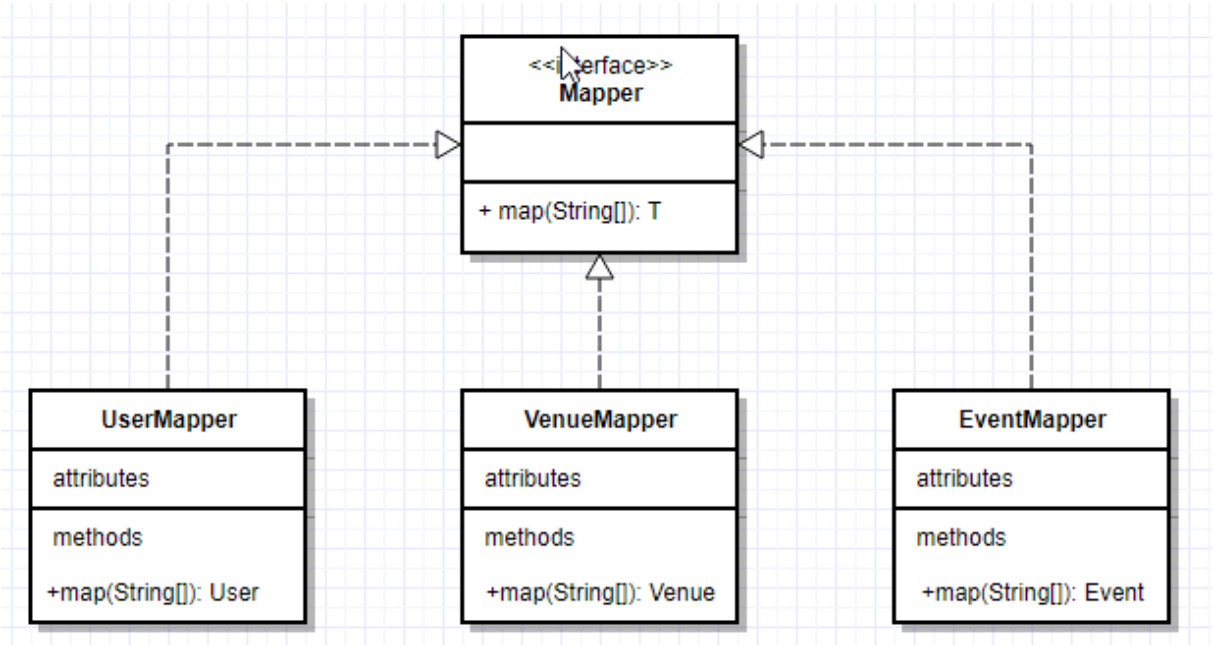
- *requestTicket(Event, User)*
  - plaatst de meegegeven User in de wachtrij van het meegegeven Event
- *viewNext(eventId)*
  - print de eerstvolgende User in de wachtrij voor het gegeven event

Uiteraard gebruik je voor bovenstaande methoden de *QueueService* waar mogelijk.

Inlezen van data

We hebben een methode nodig waarmee we automatisch data kunnen invoeren in ons ticket systeem. We hebben enkele files met data voorzien om *venues*, *events* en *users* mee aan te maken. Zorg dat er instanties van de betreffende klassen worden aangemaakt en de gegevens uit de file in deze instanties geplaatst worden.

Bij het implementeren van deze functionaliteit, mag je je baseren op onderstaande klassendiagram.



De bedoeling is dus dat je een generieke *Mapper* interface maakt. Deze heeft één abstracte methode: *map*. Deze methode krijgt data binnen als parameter en geeft een generiek type T terug. Nadien ga je deze interface implementeren in 3 afzonderlijke klassen die deze interface implementeren. Elk van deze klassen zal het generiek type T invullen als User, Event of Venue. Op die manier is er dus een *Mapper* voor elk type data en kan deze gebruikt worden om de data uit de file om te zetten in een object (User, Event of Venue).

De data bestanden zijn gewone *txt* files, je kan deze dus gerust eens bekijken met bv. *Notepad*. Je vindt de bestanden terug in de *data* folder. In de bestanden stelt elke regel telkens een aparte instantie voor.

Het formaat van deze regels, voor de verschillende types:

> **User:** *<ID>;<naam>;<voornaam>;<verjaardag (ddMMyyyy) >*
> *Voorbeeld:* U-000001;Vanderstraeten;Sam;03041987
>
> **Venue:** *<ID>;<naam>;<straatnaam>;<nummer>;<postcode>;<gemeente>;<capaciteit>*
> *Voorbeeld:* V-0001;Cirque Royale;Onderrichtstraat;81;1000;Brussel;2000
>
> **Event:** *<ID>;<tijdstip (ddMMyyyyHHmm)>;<naam>;<beschrijving>;<prijs>;<locatie ID>*
> *Voorbeeld:* RUN-00001;211120172100;Run The Jewels;World Tour;40.00;V-0001

Opmerkingen:

- Hou er rekening mee dat bijvoorbeeld een beschrijving van een event ook leeg kan zijn
- Bij het inladen van een event, moet ook de juiste *Venue* gelinkt worden aan de instantie van het *Event* (af te leiden uit het *locatie ID* in de data). Zorg zelf voor de logica om dit correct te kunnen doen.

Extra:

- Zorg dat bij het genereren van de unieke ID's van nieuwe objecten, rekening gehouden wordt met de reeds geïmporteerde data en instanties. (je zal dus de waarde van *count* moeten aanpassen)

Tickets toewijzen

Schrijf een methode *assignTickets(eventID, number)* in *TicketSystem*. Deze methode zal de eerste X personen (X = *number*) in de wachtrij van het meegegeven *event* een ticket toewijzen. Probeer zelf te achterhalen wat er precies moet gebeuren om het hele systeem up-to-date te houden. Denk hierbij bv. aan de gastenlijst van het event.

Tonen van informatie

Voorzie methoden in **TicketSystem** om onderstaande informatie te kunnen tonen (uitgeprint in console):

- events filteren op naam (bv. op titel zoeken)
- alle volgeboekte events tonen
- alle events waarvoor een bepaalde user tickets heeft
- alle locaties waar de volgende 7 dagen minstens 1 event gepland staat

Waar mogelijk, mag je gebruik maken van **streams**. Je mag eventueel methoden toevoegen of aanpassen waar je dat nodig lijkt.

Tickets genereren

Tenslotte gaan we er voor zorgen dat er ook een ticket wordt aangemaakt, op het moment dat er een ticket aan een gebruiker wordt toegewezen. We doen dat hier op een vereenvoudigde manier: we maken een tekstbestand aan met als titel *eventID_userID.txt*. In het bestand staat een opsomming van de belangrijkste gegevens die op een ticket moeten staan. Probeer hier zelf te bepalen welke gegevens er in het bestand moeten staan.

Een ticket wordt aangemaakt wanneer een gebruiker een ticket toegewezen kreeg m.b.v. de *assignTickets* methode uit het vorige onderdeel.