Java Advanced - Herhalingsoefening

In deze oefening is het jouw taak om een Java project te schrijven aan de hand van de beschrijving die hieronder gegeven wordt. Je moet zelf bepalen welke collections je best gebruikt, waar je streams gebruikt om bepaalde taken uit te voeren, etc. Uiteraard probeer je op alle plaatsen zo goed mogelijk dubbele code te vermijden. Ben je niet zeker over een bepaalde beslissing, overleg dan gerust met medestudenten of met je docent over welke oplossingen het beste zouden zijn.

Ticket systeem

In deze opdracht maak je een ticket systeem voor concertzalen. Aan elk optreden kan een locatie toegevoegd worden, waar dat optreden dus 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 op hun eigen tempo.

Creëer een **User** klasse waarmee een gebruiker kan aangemaakt worden om in het systeem tickets te boeken. Een *user* moet als member variabelen minstens de volgende waarden hebben: naam, voornaam, verjaardag (dd/MM/yyyy) en uniek ID. Het unieke ID moet steeds uit 6 cijfers bestaan, voorafgegaan door 'U-'. (bv. 'U-000003') Zorg dat de *User* klasse een constructor bevat waaraan je waarden voor al deze variabelen kan meegeven. Voorzie verder ook get- en set-functies voor elke variabele. Het unieke ID moet automatisch gegenereerd en opgeslagen worden in de constructor.

Maak een klasse **Event**. Deze klasse stelt een optreden voor van een muziekband, theatergroep, … Member variabelen voor een *Event*: naam, datum (dd/MM/yyyy), tijdstip (HH:mm), beschrijving, prijs, uniek ID. Voorzie een constructor waaraan de nodige data mee kan geven. Het unieke ID bestaat uit de eerste 3 letters van de naam van het event, gevolgd door een streepje en tenslotte een uniek getal van 5 cijfers. (bv. 'FOO-00001') Voeg ook de nodige get- en set-functies toe. Zorg dat een *Event* object ook kan bijhouden welke *Users* er een ticket gekocht hebben en dus aanwezig zullen zijn op het optreden.

Maak een klasse **Venue**. Deze klasse zal een concertzaal of -locatie voorstellen. Member variabelen voor deze klasse: naam, straatnaam, nummer, postcode, gemeente, capaciteit. Voorzie een constructor waaraan je waarden voor deze variabelen kan meegeven en de nodige get- en set-functies. Zorg ook opnieuw dat elke locatie een uniek ID krijgt: 'V-' + 4-cijferig uniek getal, bv. 'V-0007' Zorg daarna dat je een locatie kan toevoegen aan een instantie van de *Event* klasse. Deze locatie moet je kunnen zetten, veranderen, opvragen, … dus voorzie hiervoor de nodige functies.

Maak nu een **QueueService** klasse. Deze klasse moet een lijst van wachtrijen bijhouden, namelijk één voor elk event. Deze wachtrijen zullen nadien gebruikt worden om te bepalen welke gebruikers eerst een ticket toegewezen krijgen (first come, first served) Voorzie dus ook een functie *addToQueue(userID, eventID)* om gebruikers aan de wachtrij van een bepaald event toe te voegen en om de gebruikers in de juiste volgorde weer op te vragen.
Voorzie een functie *getNextInLine(eventID)* die de gebruiker teruggeeft die als volgende in de wachtrij voor een bepaald event staat en dus als volgende behandeld moet worden. Deze gebruiker moet ook automatisch verwijderd worden uit de wachtrij bij het uitvoeren van de functie.
Maak ook een functie om de wachtrij van een event uit te printen, zodat je de data later makkelijk kan

nakijken. Tenslotte moet deze klasse ook een functie bevatten om de grootte van de wachtrij van een bepaald event op te vragen.

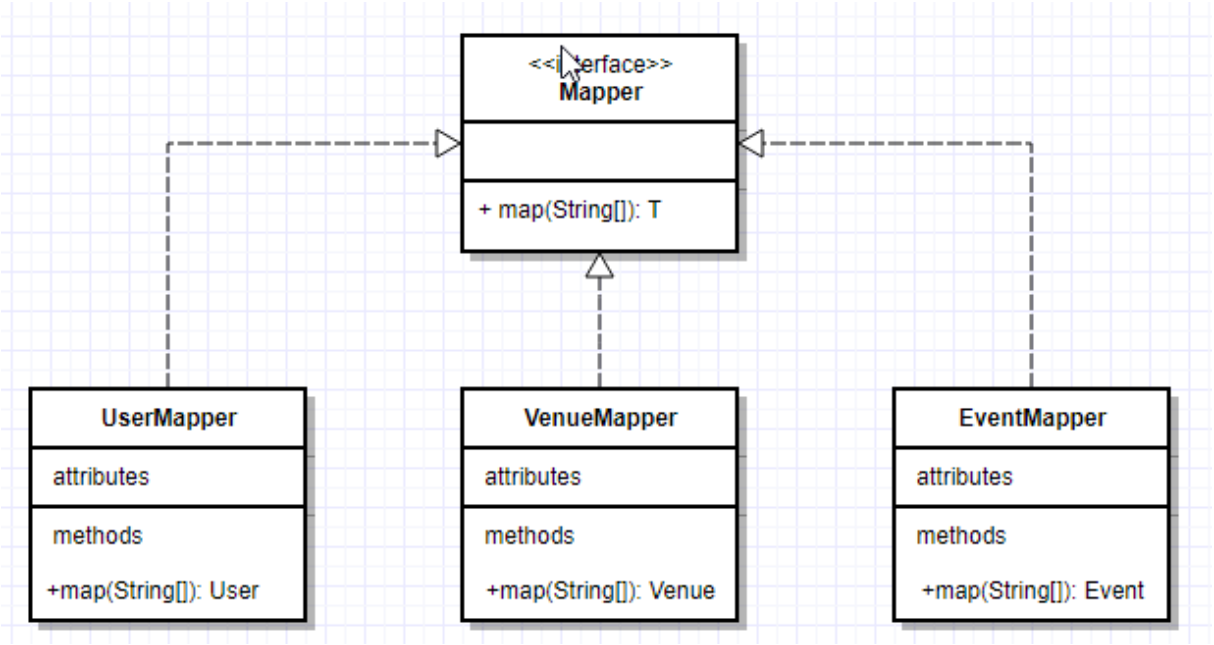
Schrijf tenslotte een klasse **TicketSystem**. Deze klasse zal al het voorgaande bundelen. In de constructor van deze klasse wordt een instantie van *QueueService* aangemaakt, die als member variabele wordt opgeslagen. Verder houdt deze klasse een lijst van users bij, een lijst van locaties en een lijst van events. Maak voor elke lijst functies om items er aan toe te voegen. (*addUser, addEvent, addVenue*) Zorg er voor dat elk soort object (users, events, …) opgevraagd kan worden aan de hand van hun unieke ID. (*getUser(id), …*) Zoek zelf naar een geschikte collection om dit te verwezenlijken.

Zorg dat er in deze klasse een functie *requestTicket()* zit. Deze functie krijgt als argumenten een *Event* en een *User* mee en zorgt dat deze user in de wachtrij gezet wordt voor een ticket voor het meegegeven event. Verder moet er ook een *viewNext()* functie zijn. *viewNext(eventID)* print de eerst volgende gebruiker in de wachtrij uit, voor het gegeven event.

Inlezen data

We hebben een functie nodig waarmee we automatisch data kunnen invoeren in ons ticket systeem. We hebben al enkele files met data voorzien om *venues*, *events* en *users* aan te maken. Zorg dat alle data correct wordt ingeladen en dat er instanties aangemaakt worden van de bijhorende klassen. Op die manier moeten we niet talloze constructors gaan toevoegen om initiële data aan te maken om bijvoorbeeld te testen.

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



De bedoeling is dus dat je een generieke *Mapper* interface maakt en daarna de data inleest met behulp van concrete klassen die deze interface implementeren.

Het formaat 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;CIR-0001

- Hou er rekening mee dat bijvoorbeeld een beschrijving van een event ook leeg kan zijn.
- Zorg dat bij het genereren van de unieke ID's van nieuwe objecten, rekening gehouden wordt met de reeds geïmporteerde data en instanties.
- 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*)

Premium users

We voorzien een functie voor gebruikers om ***premium user*** te worden. Voeg hiervoor een boolean toe in de User klasse, met bijhorende get- en set-functie. Het grote voordeel dat gebruikers krijgen als ze premium user zijn, is dat ze voorrang krijgen in de wachtrijen. Pas dus de werking binnen je *QueueService* klasse aan zodat deze hier rekening mee houdt. Indien nodig kan je de gebruikte Collection veranderen. Je zal hier waarschijnlijk ook een nieuw object voor moeten aanmaken, om het sorteren binnen de wachtrij correct te kunnen uitvoeren. Dit nieuwe object kan je bv. *TicketRequest* noemen. Bedenk zelf welke eigenschappen hier in moeten zitten, om de nodige data te voorzien.

Zorg dus dat, na je aanpassingen, de gebruikers in de wachtrijen als volgt worden afgehandeld:

- eerst degenen met premium, waarbij de gebruiker die het eerst in de wachtrij stond, als eerste afgehandeld wordt
- daarna de niet-premium gebruikers, waarbij opnieuw de eerste gebruiker eerst een ticket krijgt

Overboekingen voorkomen

Voeg een functie *isFull()* toe aan de Event klasse, die in de vorm van een boolean teruggeeft of een event volgeboekt is of niet. Zorg er ook voor dat er een exception gegooid wordt als de *addToQueue* functie wordt opgeroepen op een event dat volgeboekt is.

Overzicht optredens

Schrijf een aantal functies in *TicketSystem* om wat specifieke data te tonen i.v.m. events, locaties, … Breng zelf wijzigingen aan over de klassen waar dat volgens jou nodig is. Overleg indien je niet zeker bent.
Volgende functies moet je minimaal voorzien:

- events filteren op naam (bv. op titel zoeken)
- alle volgeboekte events tonen
- alle events waarvoor een bepaalde user in de wachtrij staat

- 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.

Tickets toewijzen

Schrijf een functie *assignTickets(eventID, number)* in *TicketSystem*. Deze functie 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 overboekingen.

Tickets aanmaken

Als uitbreiding op het vorige item, gaan we er voor zorgen dat er ook een ticket wordt aangemaakt voor gebruikers, op het moment dat er een ticket aan hen 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.