**Time:**8-16 hours

**Expected result:**show demonstration and provide code.

**Task description:**

Create Spring **application for managing a movie theater**, which allows for admins to enter events, view purchased tickets, and for users to register, view events with air dates and times, get ticket price, buy tickets.

Use the XML configuration with Spring. Configuration with annotations will be created in the next hometask. Create or reuse domain objects and services as needed.

Create DAO classes for storing data in simple static maps. In the third hometask you will replace them with classes for storing data in DB.

There is no fancy application UI for now. To demonstrate results of your work you can do one of the following:   
- create simple console application that will allow to input and output data;   
- write Spring integration unit tests (see [this documentation](https://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/testing.html));   
- create Spring shell application (see [this documentation](http://docs.spring.io/spring-shell/docs/current/reference/htmlsingle/)).

Use **Maven** for building a project. Use latest **Spring 3** version (3.x.x). Using of other build tools and Spring version should be agreed individually with course mentor/curator.

The project skeleton with domain classes, services interfaces and simple console UI implementation can be cloned from here using your EPAM username and password:

https://git.epam.com/yuriy\_tkach/spring-core-hometask-skeleton.git

Services and their descriptions that the application should provide:

**UserService** - Manages registered users

* save, remove, getById, getUserByEmail, getAll

**EventService** - Manages events (movie shows). Event contains only basic information, like name, base price for tickets, rating (high, mid, low). Event can be presented on several dates and several times within each day. For each dateTime an Event will be presented only in single Auditorium.

* save, remove, getById, getByName, getAll
* getForDateRange(from, to) - returns events for specified date range (OPTIONAL)
* getNextEvents(to) - returns events from now till the ‘to’ date (OPTIONAL)

**AuditoriumService** - Returns info about auditoriums and places

* Since auditorium information is usually static, store it in some property file. The information that needs to be stored is:
  + name
  + number of seats
  + vip seats (comma-separated list of expensive seats)

Several auditoriums can be stored in separate property files or in a single file, information from them should be injected into the AuditoriumService

* getAll(), getByName()

**BookingService** - Manages tickets, prices, bookings

* getTicketsPrice(event, dateTime, user, seats) - returns total price for buying all tickets for specified event on specific date and time for specified seats.
  + User is needed to calculate discount (see below)
  + Event is needed to get base price, rating
  + Vip seats should cost more than regular seats (For example, 2xBasePrice)
  + All prices for high rated movies should be higher (For example, 1.2xBasePrice)
* bookTicket(tickets) - Ticket should contain information about event, air dateTime, seat, and user. The user could be registered or not. If user is registered, then booking information is stored for that user (in the tickets collection). Purchased tickets for particular event should be stored.
* getPurchasedTicketsForEvent(event, dateTime) - get all purchased tickets for event for specific date and Time

**DiscountService** - Counts different discounts for purchased tickets

* getDiscount(user, event, dateTime, numberOfTickets) - returns total discount (from 0 to 100) that can be applied to the user buying specified number of tickets for specific event and air dateTime
* DiscountStrategy - single class with logic for calculating discount
  + - Birthday strategy - give 5% if user has birthday within 5 days of air date
    - Every 10th ticket - give 50% for every 10th ticket purchased by user. This strategy can also be applied for not-registered users if 10 or more tickets are bought
* All discount strategies should be injected as list into the DiscountService. The getDiscount method will execute each strategy to get max available discount. Discounts should not add up. So, if one strategy returns 20% and another 30%, final discount would be 30%.
* Define DiscountService with all strategies as separate beans in separate XML configuration

**A note for creativity**. If you feel that assignment lacks some specific details then you must know that it was done intentionally, so you can be creative and enrich/implement it in your own way. You are free to add some fields/methods to domain classes or services, which you think will help in accomplishing your goal for creating really cool movie theater managing application.

**How to present your work**.

You can create your own project at [https://git.epam.com](https://git.epam.com/) and commit/push there. After that provide your tutor/mentor with the project url for checking the work done. Alternatively, you can zip your sources and send them by e-mail or other means. In any case, you have to agree with your mentor/curator on how the work would be submitted.