

Meeting calendar

Topics:

- Object oriented programming
- Composition
- Aggregation
- ArrayList
- UML diagrams
- Design patterns
- Sequencers

Overview:

You are going to create a meeting calendar project where you create and define objects of type Person, AppUser and Meeting. Additionally, you need to create a central storage (DataAccessObject) for each model (Person, AppUser and Meeting) type.

Requirements:

- AppUser, Person and Meeting **fully implemented and tested** according to specific requirements.
- Sequencers **fully implemented**.
- AppUserRepository, PersonRepository and MeetingRepository **fully implemented** according to specific requirements.
- (If you have time) AppUserService, PersonService and MeetingService **fully implemented and tested** according to specific requirements.

Step 1 – Creation of AppUser, Person and Meeting.

First create a new package called **models**.

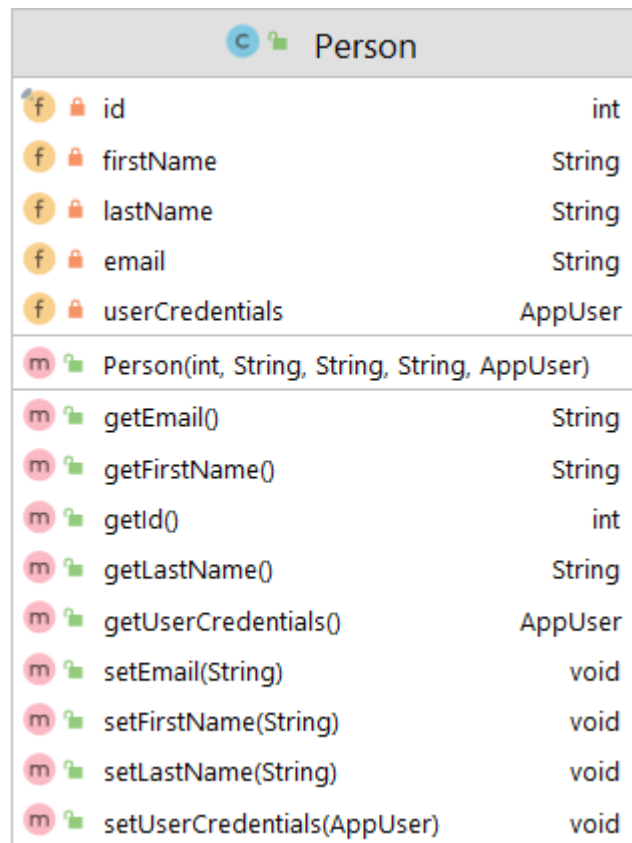
Create AppUser class:

1. Inside models package create AppUser class according to diagram.
2. Username and password are **not allowed to be null or empty**.
3. At object instantiation send in generated id from AppUserSequencer. (See sequencer specifics)
4. (Optional) Generate equals and hashCode.
5. Test AppUser with Junit 5 or 4.

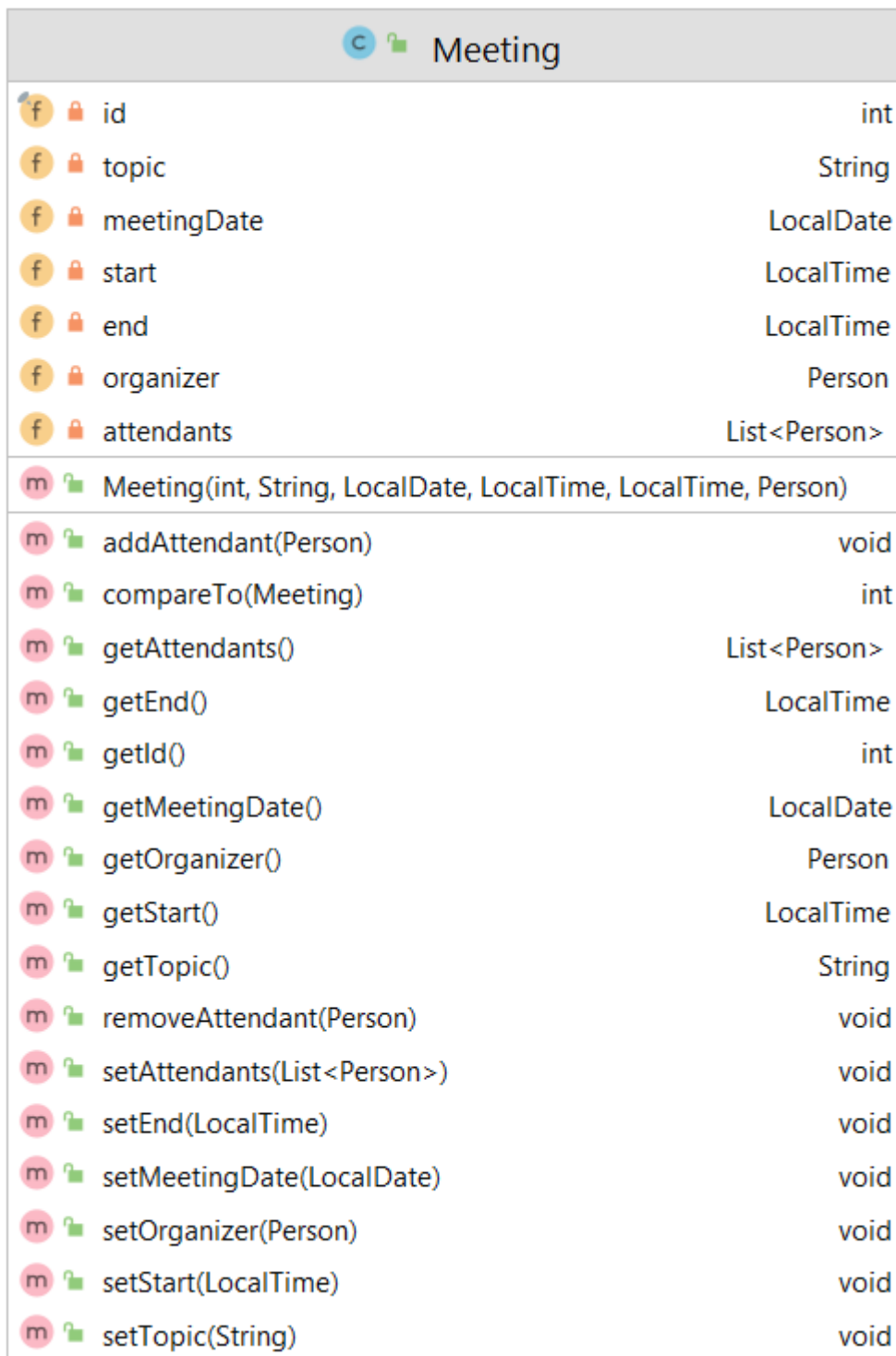
AppUser		
f	id	int
f	username	String
f	password	String
m	AppUser(int, String, String)	
m	getId()	int
m	getPassword()	String
m	getUsername()	String
m	setPassword(String)	void
m	setUsername(String)	void

Create Person class

1. Inside models package create Person class according to diagram.
2. FirstName, lastName, email, userCredentials are **not allowed to be null or empty**.
3. At object instantiation send in generated id from PersonSequencer. (See sequencer specifics)
4. (Optional) Generate equals and hashCode excluding userCredentials.
5. Test Person with Junit 5 or 4.

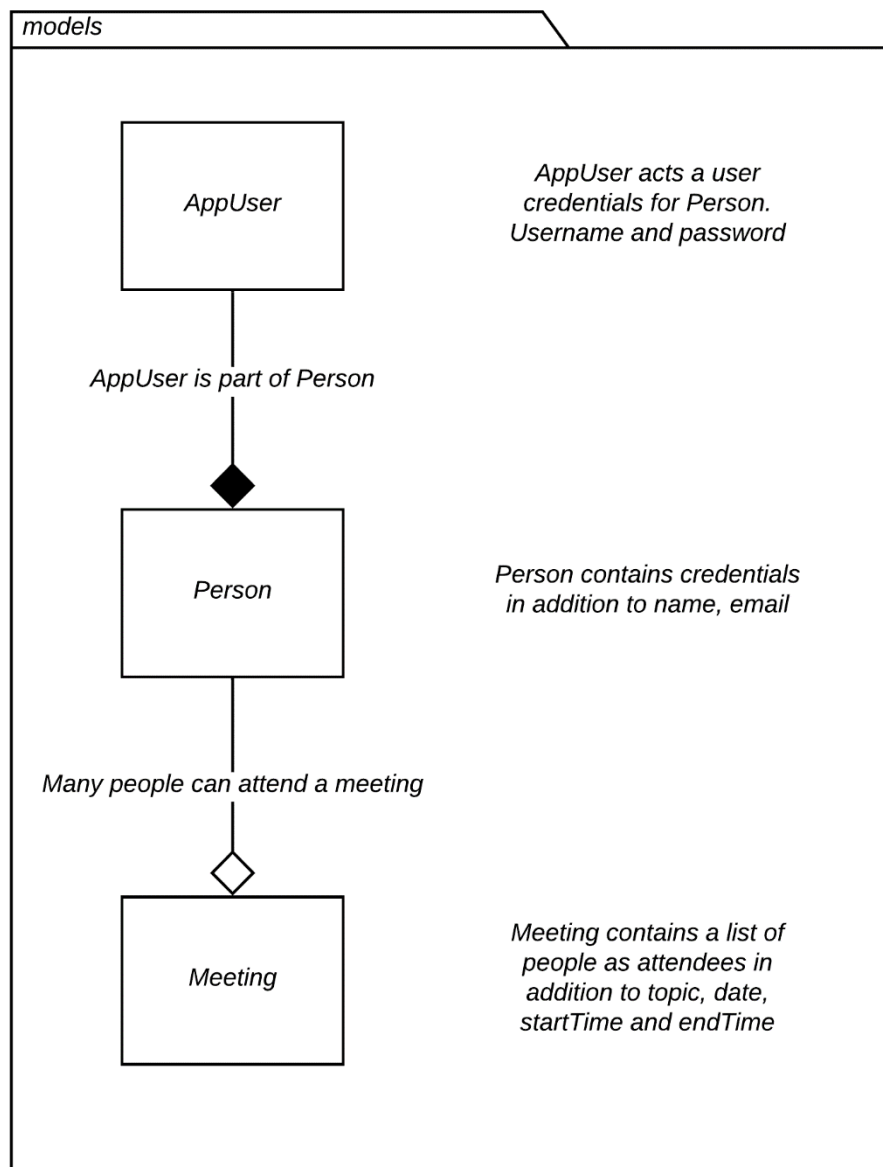


Create Meeting class



1. Inside models create Meeting class according to diagram.
2. Organizer, topic, meetingDate, start, end are **not allowed to be null**
3. At object instantiation send in generated id from MeetingSequencer.
4. Test with Junit 5 or 4
5. (Optional) Generate equals and hashCode excluding attendants and organizer.

Model package relationships overview:



Step 2 – Creation of Sequencers

Background:

When dealing with numeric id's like int we need to create classes that keeps track on what id's has been assigned here is where sequencers come in handy.

First you should create a **new package sequencers**

Create AppUserSequencer according to description.

Note that **appUserId** is a **static int** and **nextAppUserId()** , **reset()** are **static methods**.

(If you choose to make appUserId non static you need to make AppUserSequencer into a Singleton.)

Create MeetingSequencer like you did with AppUserSequencer.

Create PersonSequencer like you did with the other sequencers.

AppUserSequencer		
f	appUserId	int
m	nextAppUserId()	int
m	reset()	void

MeetingSequencer		
f	meetingId	int
m	nextMeetingId()	int
m	reset()	void

PersonSequencer		
f	personId	int
m	nextPersonId()	int
m	reset()	void

Step 3 – Creation of central storage of AppUser, Person and Meeting















Description:





















These classes have the functionality of storing objects and retrieving them from an ArrayList.
(Normally a database, but we will get there in time...)























Requirements:

- Create a package named **data**.
- Inside data package create the classes according to the **specifications below**.
- (Optional but recommended) Make all classes inside package data into a **singletons**.
- You are **forbidden to use static** in any of these classes except where needed (Like when making them into a **singleton**)
- If you have made your classes into a singleton then add a *public void clear()* (much needed in testing) method inside each created class.
- (Optional) Test all methods inside AppUserRepository, PersonRepository and MeetingRepository.

AppUserRepository, PersonRepository and MeetingRepository:

AppUserRepository		
 	appUserStorage	List<AppUser>
 	findAll()	List<AppUser>
 	findById(int)	AppUser
 	findByUsername(String)	AppUser
 	getAppUserCount()	int
 	persist(AppUser)	AppUser
 	remove(int)	boolean

PersonRepository		
 	peopleStorage	List<Person>
 	findAll()	List<Person>
 	findByEmail(String)	Person
 	findByFirstName(String)	List<Person>
 	findById(int)	Person
 	findByLastName(String)	List<Person>
 	findByUsername(String)	Person
 	getPeopleCount()	int
 	persist(Person)	Person
 	remove(int)	boolean

MeetingRepository		
 	meetings	List<Meeting>
 	findAll()	List<Meeting>
 	findByAttendeePersonId(int)	List<Meeting>
 	findById(int)	Meeting
 	findByMeetingDate(LocalDate)	List<Meeting>
 	findByMeetingsBetween(LocalDateTime, LocalDateTime)	List<Meeting>
 	findByOrganizerPersonId(int)	List<Meeting>
 	findByTopic(String)	List<Meeting>
 	getMeetingCount()	int
 	persist(Meeting)	Meeting
 	remove(int)	boolean

Step 4 (If you have time) – Create a service layer:

Description:

A Service layer typically contains a lot of business logic. You are recommended to make each of the service classes into a singleton.

Method descriptions:

AppUser create(String username, String password)

Create is responsible for instantiating an AppUser object. Before instantiating AppUser make sure that String username is not already present in appUserRepository. Use AppUserSequencer class to get a unique id for AppUser.

Update is going to fetch AppUser by id and update found object with new username and/or password. If updating username make sure that new username is not found in the database.

All the find methods are just delegations from the appUserRepository and can be generated by IntelliJ

You are required to **test each method** inside **AppUserService**.

AppUserService		
f	appUserRepository	AppUserRepository
m	create(String, String)	AppUser
m	findAll()	List<AppUser>
m	findById(int)	AppUser
m	findByUsername(String)	AppUser
m	remove(int)	boolean
m	update(int, String, String)	AppUser

Method descriptions:

Create is responsible for instantiating Person. It takes five parameters: **username, password, firstName, lastName and email**. Username and email need to be unique. You are also **not allowed** to instantiate a Person object that does not contains its userCredentials.

Update takes four parameters: **id, firstName, lastName and email**. Apply similar update logic as you did in AppUserService. Remember to check email before updating the field.

All the find methods are just delegations from the personRepository and can be generated by IntelliJ

You are required to **test each method** inside **PersonService**.

PersonService		
f	personRepository	PersonRepository
f	appUserService	AppUserService
m	create(String, String, String, String, String)	Person
m	findAll()	List<Person>
m	findByEmail(String)	Person
m	findByFirstName(String)	List<Person>
m	findById(int)	Person
m	findByLastName(String)	List<Person>
m	remove(int)	boolean
m	update(int, String, String, String)	Person

MeetingService		
f	meetingRepository	MeetingRepository
f	personService	PersonService
m	addAttendant(int, int)	Meeting
m	create(String, LocalDateTime, LocalDateTime, String)	Meeting
m	create(String, LocalDateTime, LocalDateTime, String, List<Integer>)	Meeting
m	findAll()	List<Meeting>
m	findByAttendeePersonId(int)	List<Meeting>
m	findById(int)	Meeting
m	findByMeetingDate(LocalDate)	List<Meeting>
m	findByMeetingsBetween(LocalDateTime, LocalDateTime)	List<Meeting>
m	findByOrganizerPersonId(int)	List<Meeting>
m	findByTopic(String)	List<Meeting>
m	remove(int)	boolean
m	removeAttendant(int, int)	Meeting
m	update(int, String, LocalDate, LocalTime, LocalTime)	Meeting

Method descriptions:

This class has two overloaded create methods. Both methods have these parameters: **topic, start, end and username** (for the organizer).

The **List of integers** in the second create method contains id's for meeting attendants that should be added.

addAttendant and **removeAttendant** has the same parameters: **meetingId** and **personId**

All the find methods are just delegations from the meetingRepository and can be generated by IntelliJ

You are required to **test each method** inside **MeetingService**.

Overview of service and data layer

