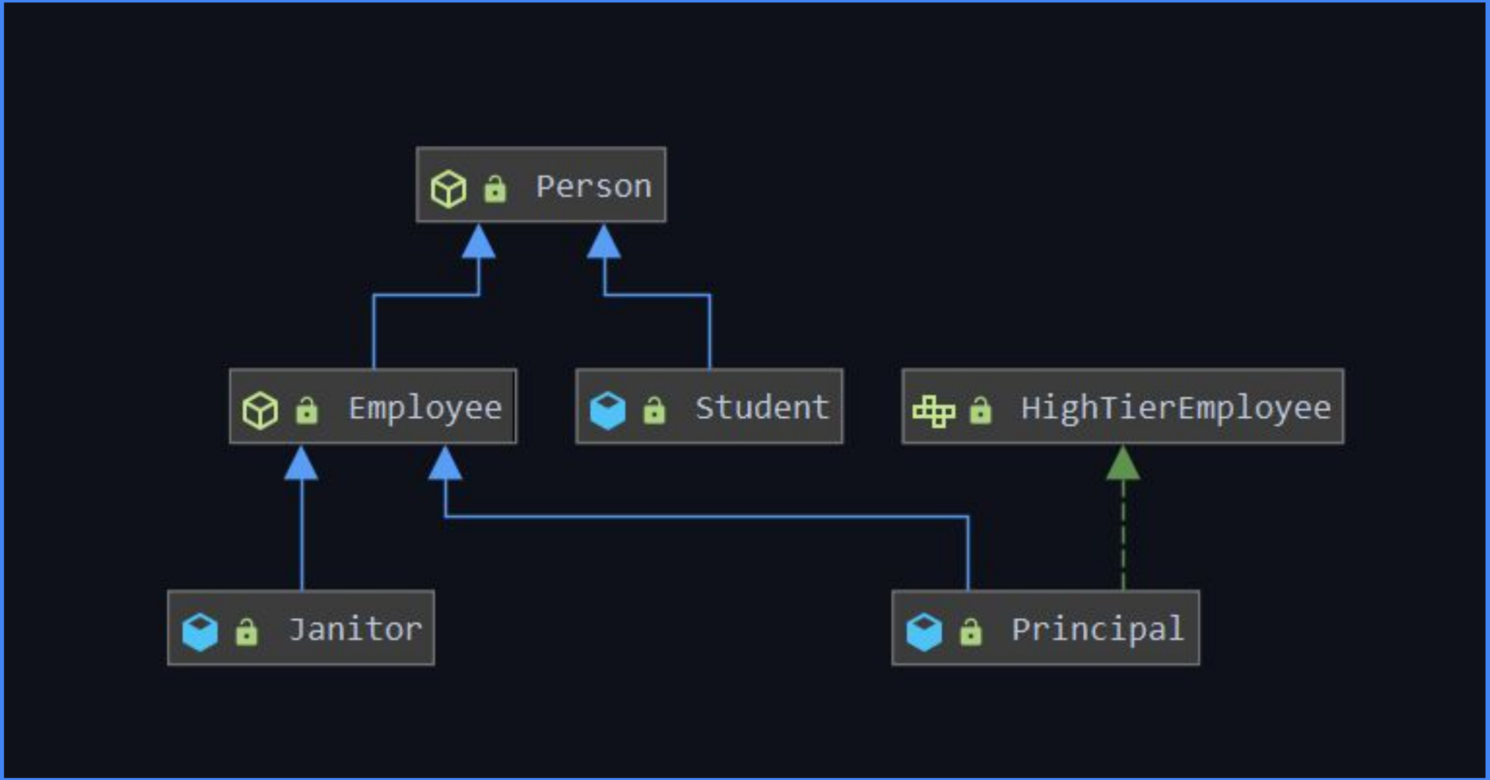
Developer hiring exam 2019

Answers

Exercise 1

A) Diagrams

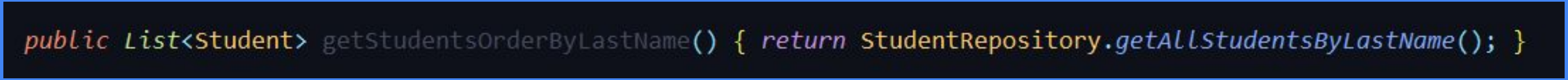


= Abstract

= Class

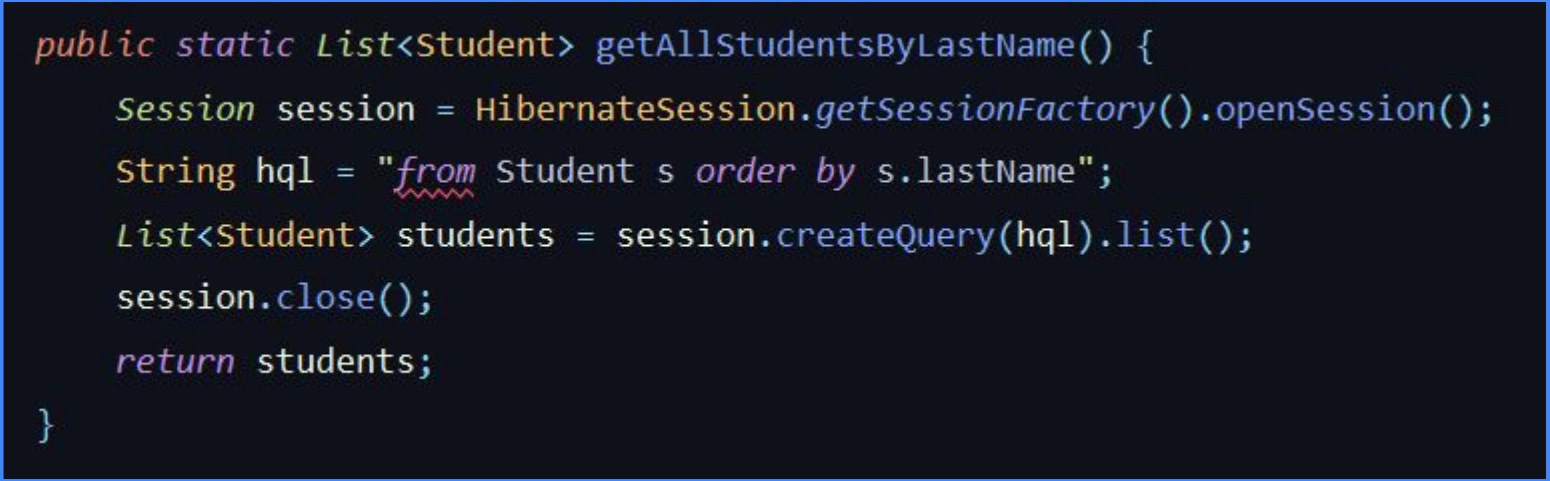
= Interface

B) Students by last name



```
public List<Student> getStudentsOrderByLastName() { return StudentRepository.getAllStudentsByLastName(); }
```

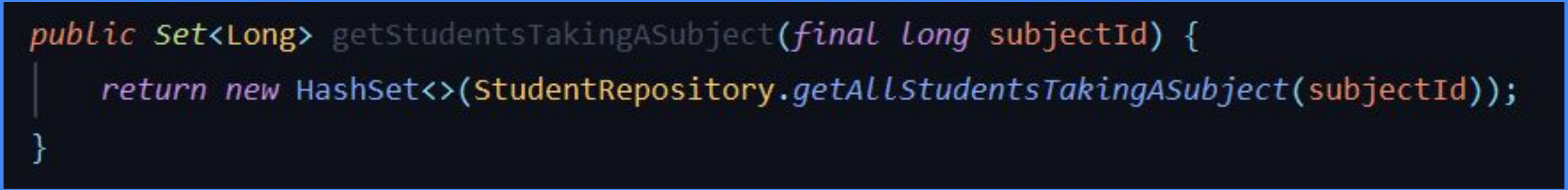
https://github.com/MaggieMarchena/developer-hiring-exam-2019/blob/master/src/main/java/exercise1/service/StudentService.java#L11



```
public static List<Student> getAllStudentsByLastName() {
    Session session = HibernateSession.getSessionFactory().openSession();
    String hql = "from Student s order by s.lastName";
    List<Student> students = session.createQuery(hql).list();
    session.close();
    return students;
}
```

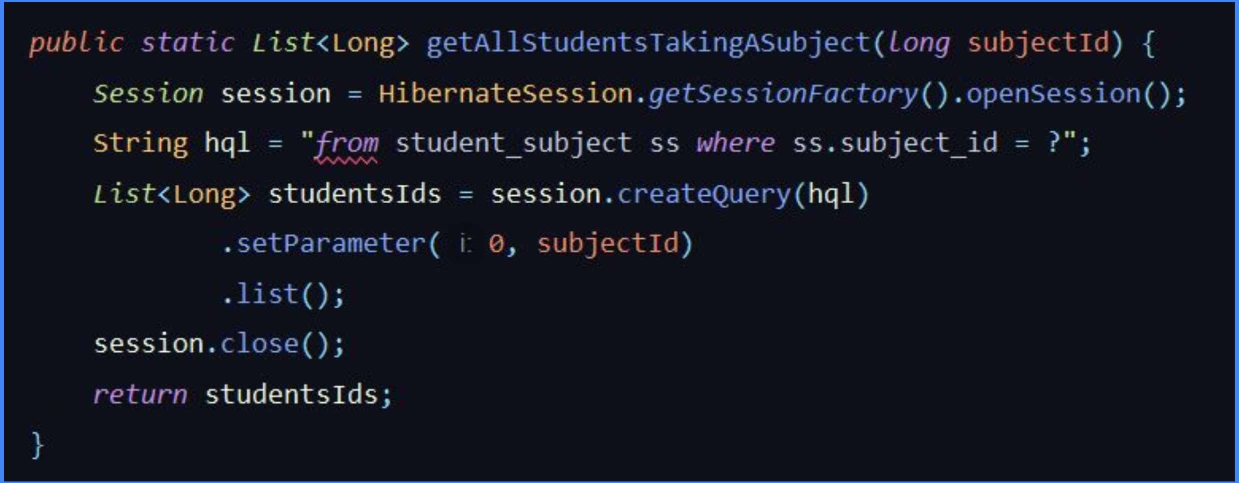
https://github.com/MaggieMarchena/developer-hiring-exam-2019/blob/master/src/main/java/exercise1/repository/StudentRepository.java#L22

C) Students taking a subject



```java
public Set<Long> getStudentsTakingASubject(final long subjectId) {
    return new HashSet<>(StudentRepository.getAllStudentsTakingASubject(subjectId));
}
```

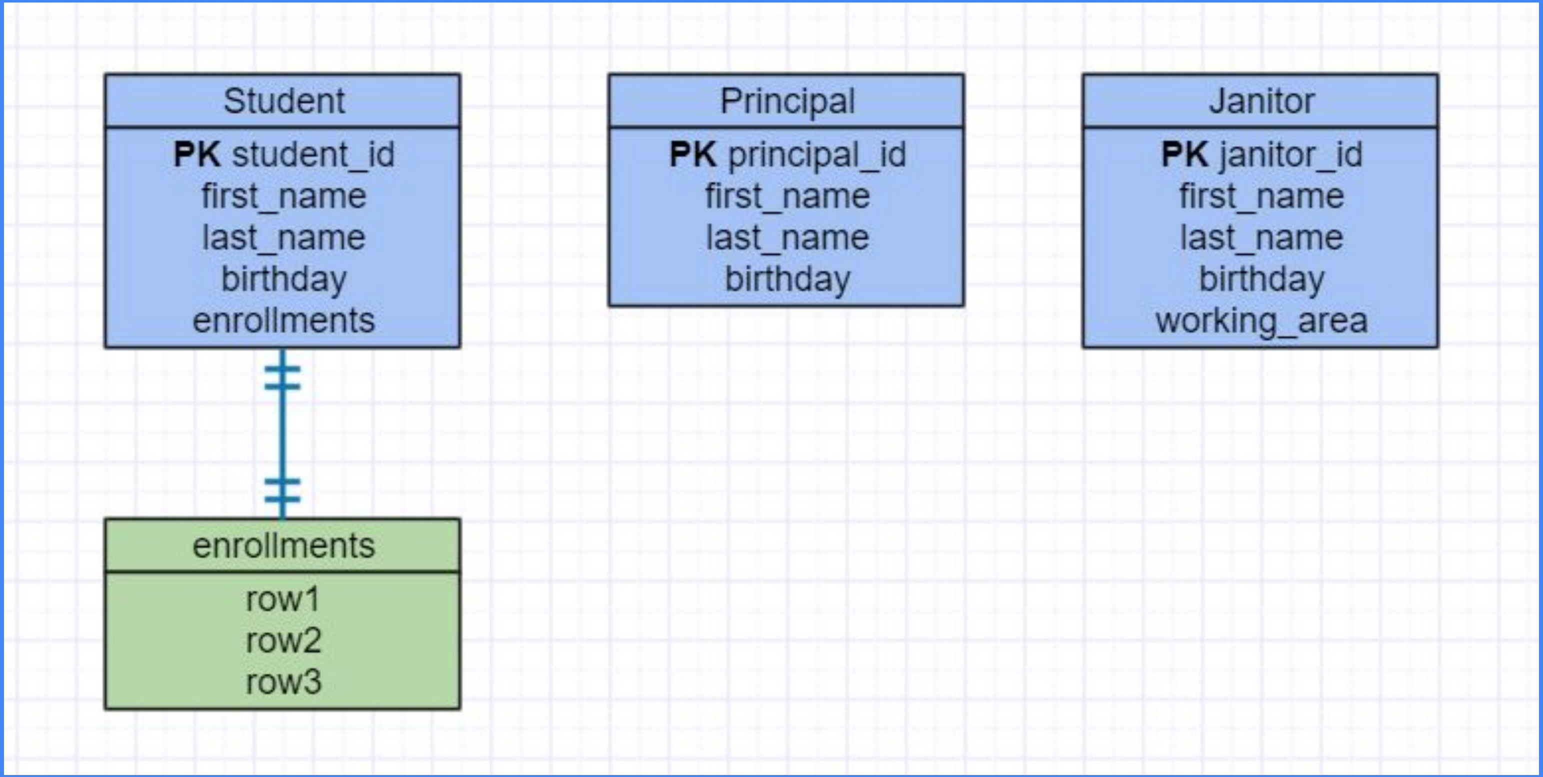
https://github.com/MaggieMarchena/developer-hiring-exam-2019/blob/master/src/main/java/exercise1/service/StudentService.java#L15



```java
public static List<Long> getAllStudentsTakingASubject(long subjectId) {
    Session session = HibernateSession.getSessionFactory().openSession();
    String hql = "from student_subject ss where ss.subject_id = ?";
    List<Long> studentsIds = session.createQuery(hql)
            .setParameter( i: 0, subjectId)
            .list();
    session.close();
    return studentsIds;
}
```

https://github.com/MaggieMarchena/developer-hiring-exam-2019/blob/master/src/main/java/exercise1/repository/StudentRepository.java#L12

D) Database option 1



D) Database option 1

In this initial approach enrollments is a List<String> and is saved to the database as a Collection with the @ElementCollection annotation which creates an additional table for the collection.

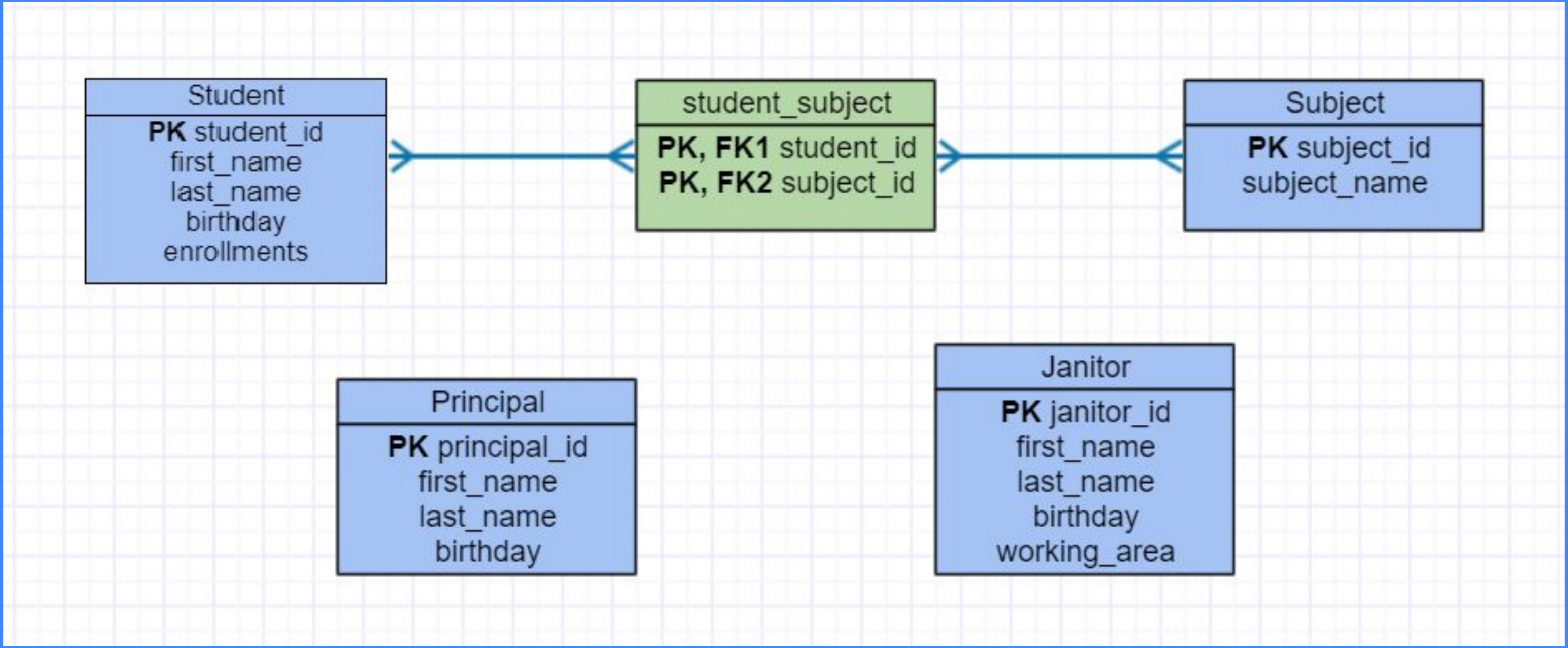
PROS:

- Fewer entities to handle.
- Simpler structure to code.
- Easier ways to return data from the list.

CONS:

- Collection elements aren't persisted with an ID so they can't be treated separately.
- To add or remove an object, the whole collection is wiped and created again with the change making it un-efficient.

D) Database option 2



D) Database option 2

In this second approach the class Subject is created and enrollments becomes a List<Subject> so the relationship rows are persisted under a new student_subject table.

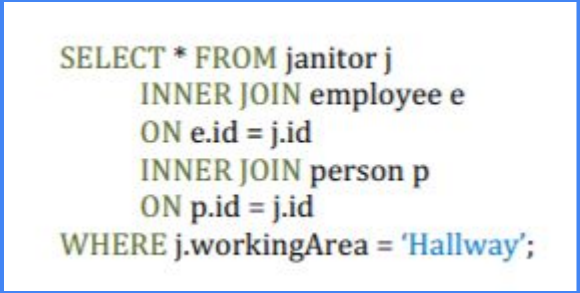
PROS:

- Student_subject relationship can be queried.
- More efficient.

CONS:

- More complex queries.
- More entities to handle.
- Harder to code

E) Optimizing query/database



```
SELECT * FROM janitor j
        INNER JOIN employee e
        ON e.id = j.id
        INNER JOIN person p
        ON p.id = j.id
WHERE j.workingArea = 'Hallway';
```

- Select only the fields you need j.first_name and j.last_name
- Eliminate ‘Person’ and ‘Employee” and keep the data in ‘janitors’ table

```
SELECT j.first_name, j.last_name
FROM janitors j
WHERE j.working_area = “Hallway”
```

F) Optimizing querying

Create a table with the data involved in the report and use it for the repetitive query.

Then create a method/service to update this table whenever the original tables change.

G) Students between 19-21 query

I don’t know how to make this query but from the service I would:

- Get all students
- Stream and filter to match:

  birthday >= currentDate - 19 && birthday <= currentDate - 21

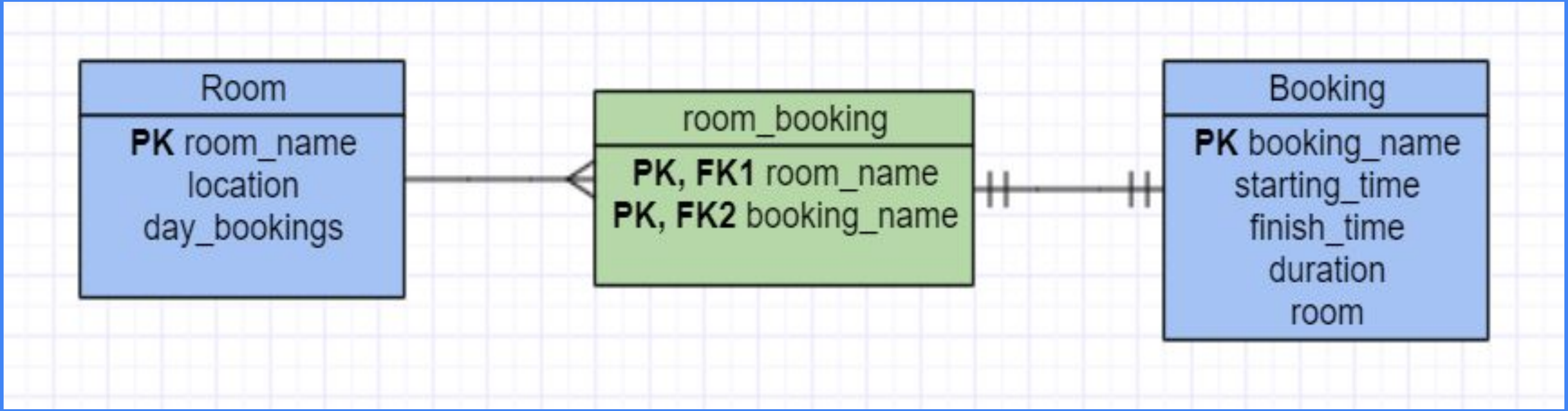
H) Business logic in database

- Check validations through triggers on INSERT

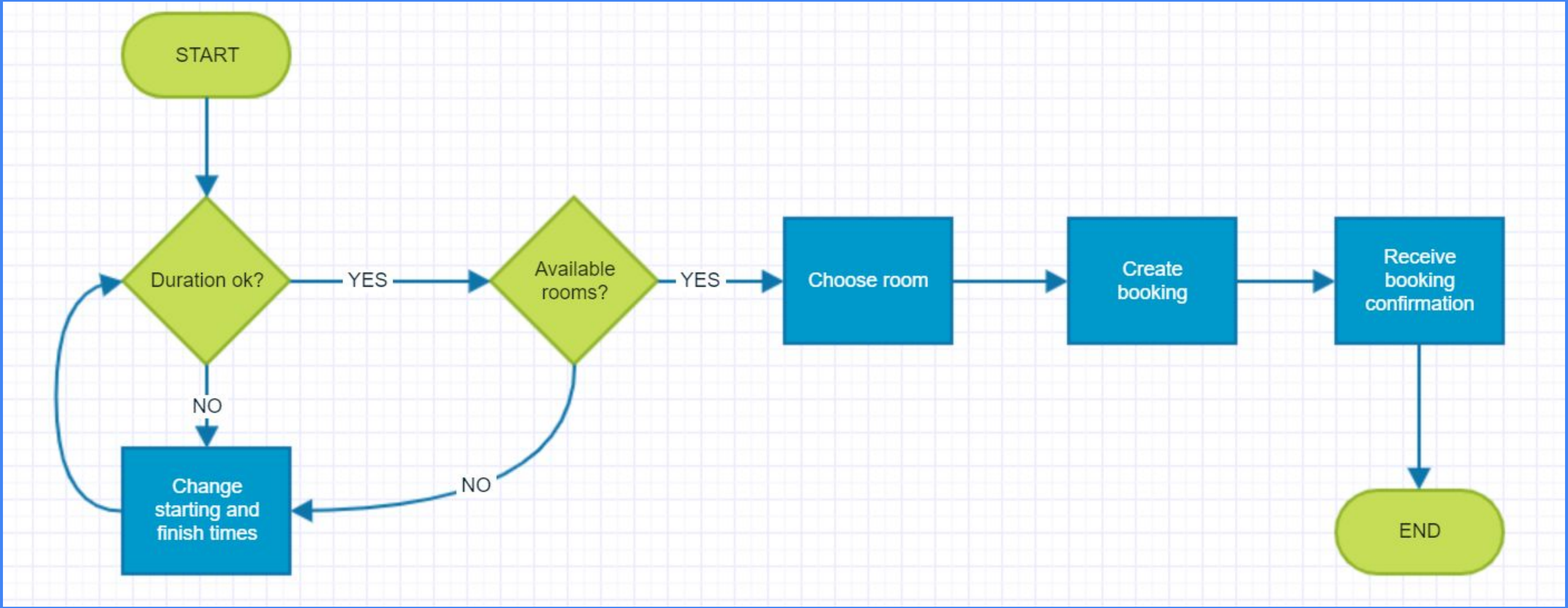
On the pros side, having the logic within the database will make the app faster but it will result in a more complex code and slower database

Exercise 2

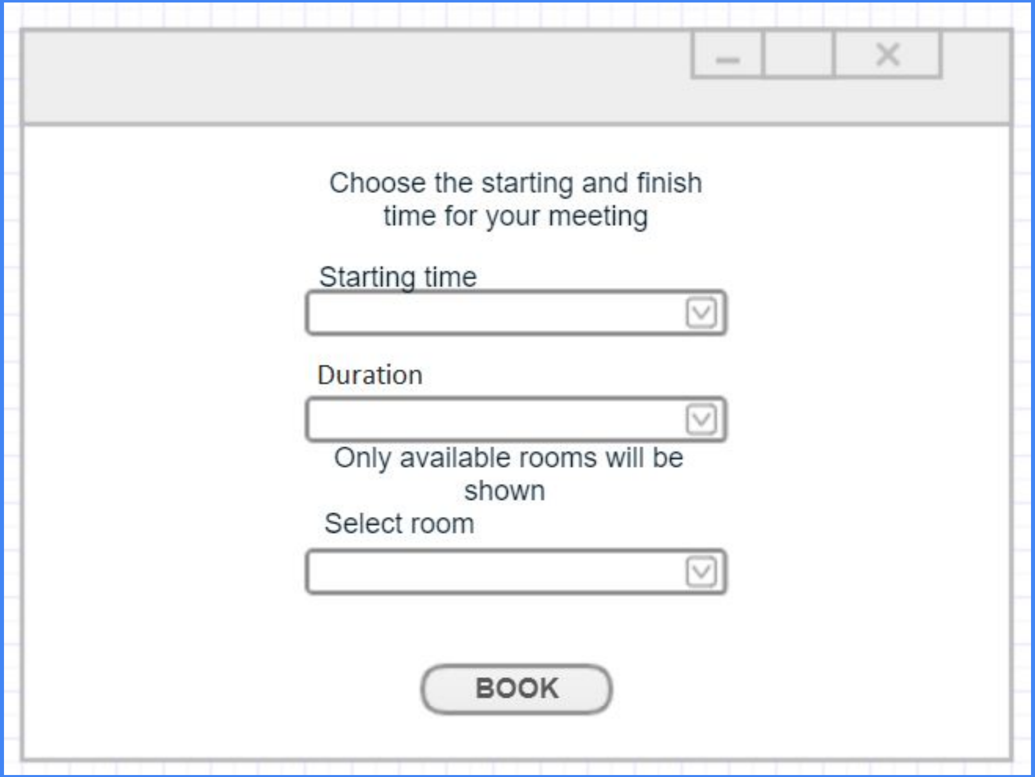
A) Entity diagram



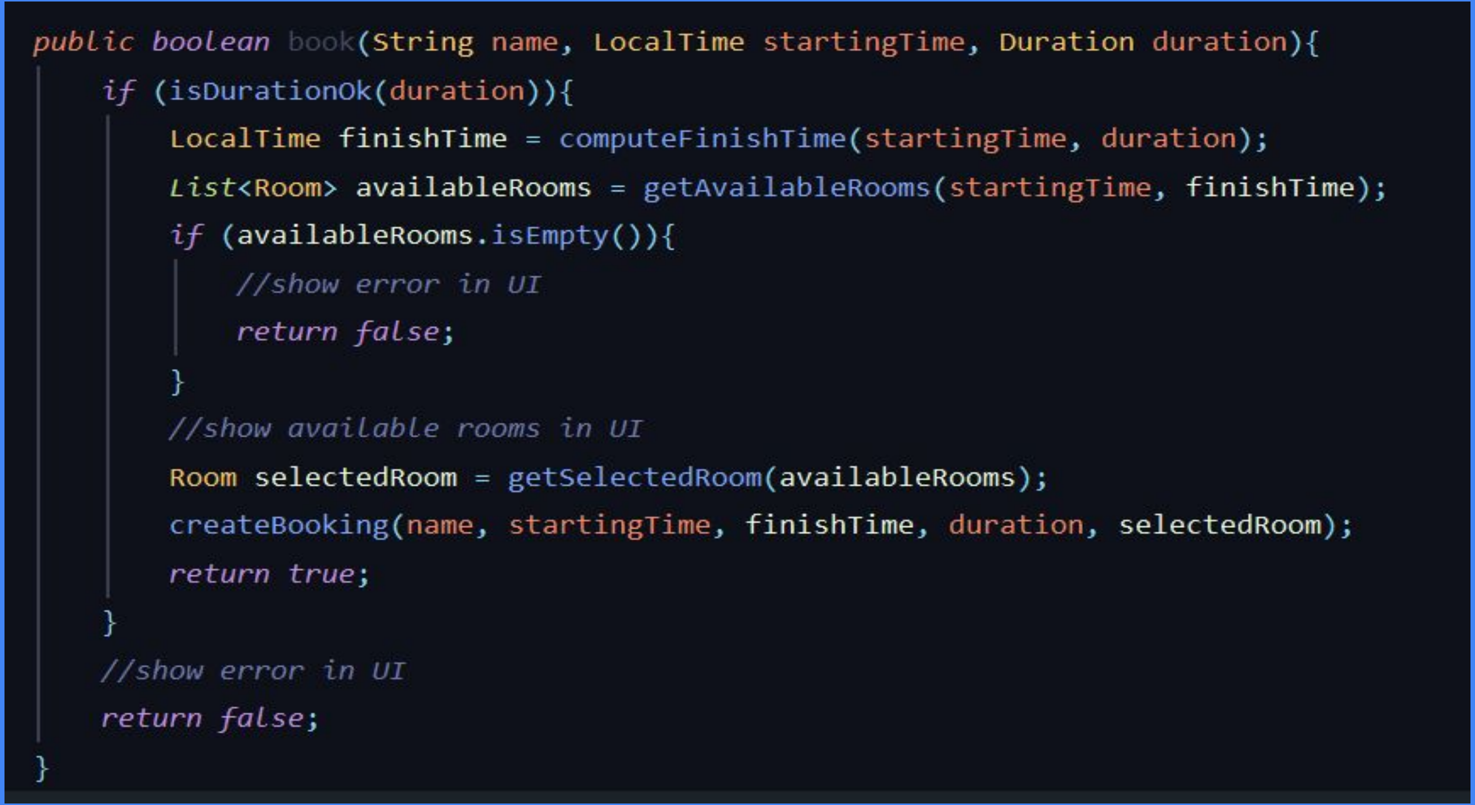
B) User interface diagram



B) User interface diagram


Choose the starting and finish time for your meeting
Starting time
Duration
Only available rooms will be shown
Select room
BOOK

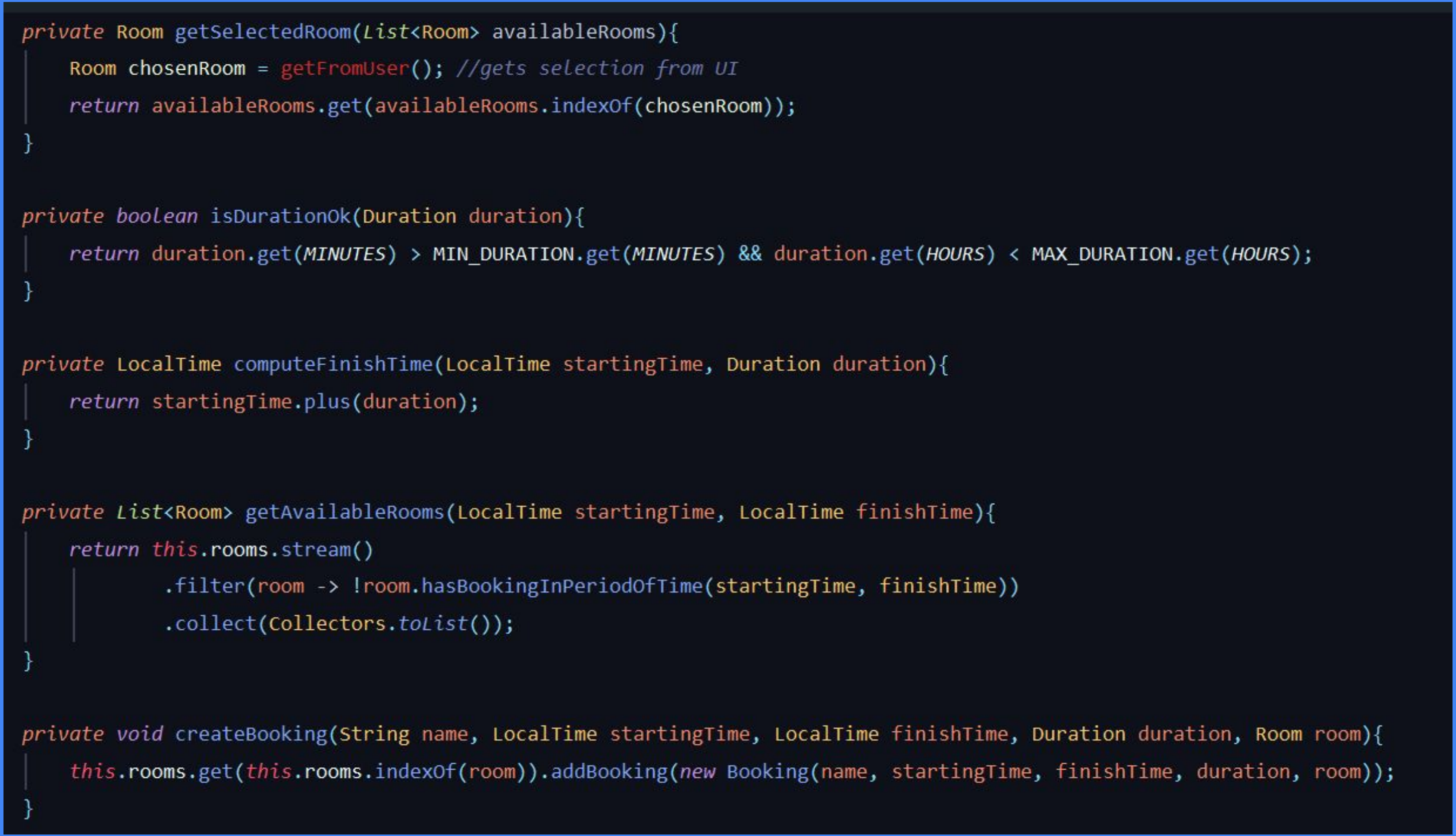
C)Booking method



```java
public boolean book(String name, LocalTime startingTime, Duration duration){
    if (isDurationOk(duration)){
        LocalTime finishTime = computeFinishTime(startingTime, duration);
        List<Room> availableRooms = getAvailableRooms(startingTime, finishTime);
        if (availableRooms.isEmpty()){
            //show error in UI
            return false;
        }
        //show available rooms in UI
        Room selectedRoom = getSelectedRoom(availableRooms);
        createBooking(name, startingTime, finishTime, duration, selectedRoom);
        return true;
    }
    //show error in UI
    return false;
}
```

https://github.com/MaggieMarchena/developer-hiring-exam-2019/blob/master/src/main/java/exercise2/service/BookingService.java#L32

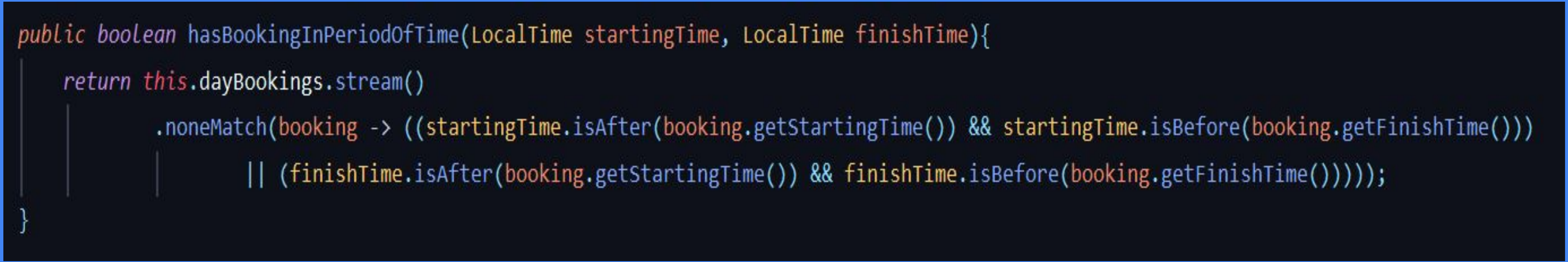
C)Booking checks



```java
private Room getSelectedRoom(List<Room> availableRooms){
    Room chosenRoom = getFromUser(); //gets selection from UI
    return availableRooms.get(availableRooms.indexOf(chosenRoom));
}

private boolean isDurationOk(Duration duration){
    return duration.get(MINUTES) > MIN_DURATION.get(MINUTES) && duration.get(HOURS) < MAX_DURATION.get(HOURS);
}

private LocalTime computeFinishTime(LocalTime startingTime, Duration duration){
    return startingTime.plus(duration);
}

private List<Room> getAvailableRooms(LocalTime startingTime, LocalTime finishTime){
    return this.rooms.stream()
            .filter(room -> !room.hasBookingInPeriodOfTime(startingTime, finishTime))
            .collect(Collectors.toList());
}

private void createBooking(String name, LocalTime startingTime, LocalTime finishTime, Duration duration, Room room){
    this.rooms.get(this.rooms.indexOf(room)).addBooking(new Booking(name, startingTime, finishTime, duration, room));
}
```

https://github.com/MaggieMarchena/developer-hiring-exam-2019/blob/master/src/main/java/exercise2/service/BookingService.java#L49

C)Room availability check



```java
public boolean hasBookingInPeriodOfTime(LocalTime startingTime, LocalTime finishTime){
    return this.dayBookings.stream()
            .noneMatch(booking -> ((startingTime.isAfter(booking.getStartingTime()) && startingTime.isBefore(booking.getFinishTime()))
                    || (finishTime.isAfter(booking.getStartingTime()) && finishTime.isBefore(booking.getFinishTime()))));
}
```

https://github.com/MaggieMarchena/developer-hiring-exam-2019/blob/master/src/main/java/exercise2/model/Room.java#L23