KATALYST

Corporate Hotel Booking

Difficulty: Expert

Estimated Duration: 5 hours

CORPORATE HOTEL BOOKING KATA

created by Sandro Mancuso

Build a corporate hotel booking engine. This engine has to satisfy the needs of 3 different types of actors:

- **Hotel Manager:** Set all the different types of rooms and respective quantities for her hotel.
- **Company Admin:** Add/delete employees and also create booking policies for her company and employees.
- **Employee:** Book a hotel room

To achieve that, the engine needs to provide 4 main services that work in close collaboration with each other.

The four services are described below. The `<?>` indicates you can use whatever primitive or type you want.

Hotel Service

Used by the hotel manager to define the types and number of rooms of each type the hotel has. It also can return hotel information given a hotel ID.

```
public class HotelService {

    // Collaborators(?)

    void addHotel(<?> hotelId, <?> hotelName);

    void setRoom(<?> hotelId, <?> number, <?> roomType);

    <?> findHotelBy(<?> hotelId);

}
```

Rules

The `addHotel(...)` method should throw an exception when the hotel ID already exists or create the hotel otherwise.

The `setRoom(...)` method should throw an exception if the hotel does not exist. It should insert or update a room according to its room number.

The `findHotelBy(<?> hotelId)` should return all the information about the number of rooms for the specified ID.

Company Service

Enables company admins to add and delete employees.

```
public class CompanyService {

    // Collaborators(?)

    void addEmployee(<?> companyId, <?> employeeId);

    void deleteEmployee(<?> employeeId);

}
```

Rules

- Employees should not be duplicated.
- When deleting an employee, all the bookings and policies associated to the employee should also be deleted from the system.

Booking Policy Service

Allows company admins to create booking policies for their company and employees. Booking policies determine if an employee booking request is allowed by their company. There are two types of booking policy:

- Company Booking Policy: Indicates which type of rooms can be booked. E.g. a company may only allow standard (single/double) rooms to be booked. Or it may allow standard and junior suite rooms.
- Employee Booking Policy: Indicates which type of rooms a specific employee can book. E.g. One employee might only be allowed to book a standard room while another employee may be allowed to book standard, junior suite and master suite.

```
public class BookingPolicyService {

    // Collaborators(?)

    void setCompanyPolicy(<?> companyId, <?> roomTypes);

    void setEmployeePolicy(<?> employeeId, <?> roomTypes);

    boolean isBookingAllowed(<?> employeeId, <?> roomType);

}
```

Business Rules

- Employee policies take precedence over company policies. If there is a policy for an employee, the policy should be respected regardless of what the company policy (if any) says.
- If an employee policy does not exist, the company policy should be checked.
- If neither employee nor company policies exist, the employee should be allowed to book any room.

Technical Rules

- Methods `setCompanyPolicy(...)` and `setEmployeePolicy(...)` should create a new policy or update an existing one. No duplicate company or employee policies are allowed.
- Method `isBookingAllowed(...)` should take into account the employee and the company the employee works for.

Booking Service

Allows employees to book rooms at hotels.

```
public class BookingService {

    // Collaborators (?)

    Booking book(<?> employeeId, <?> hotelId, <?> roomType, Date checkIn, Date checkOut);

}
```

Rules

- Booking should contain a unique ID, employeeId, hotelId, roomType, checkIn and checkOut.

- Check out date must be at least one day after the check in date.
- Validate if the hotel exists and room type is provided by the hotel
- Verify if booking is allowed according to the booking policies defined, if any. See Booking Policy Service for more details.
- Booking should only be allowed if there is at least one room type available during the whole booking period.
- Keep track of all bookings. E.g. If hotel has 5 standard rooms, we should have no more than 5 bookings in the same day.
- Hotel rooms can be booked many times as long as there are no conflicts with the dates.
- Return booking confirmation to the employee or error otherwise (exceptions can also be used).

Kata guidelines

- Outside-In TDD should be used.
- Starting from acceptance tests, one at a time.
- Define your acceptance tests as small vertical slices, involving at least two services.
- Once a failing acceptance test is compiling and failing for the right reasons (services are not implemented), start unit testing the services, one public method at a time.

Design guidelines

- The public interface of the services cannot be changed, except their constructors in order to inject collaborators.
- <?> can be replaced with whichever type or primitive you prefer.
- Services should not have circular dependencies, that means if service A uses B, service B cannot use service A.
- Services should not have any state.
- Persistent data should be stored in a (in-memory) repository.
- Create a good package structure, respecting the different users and areas of the application.

Kata variations

Bellow are a few variations of this kata that you can mix and match.

1. Defined Services, undefined public interfaces

In this variation you still need to use the 4 services defined above, but are totally free to create whichever methods you like in each one of the services.

2. Undefined initial design (Advanced)

In this variation, all the business requirements for the 3 different actors remain the same, but you are free to design the way you like. No need to stick to the 4 services.

3. Hotel booking policies

In the original requirements, only room types are taken into account. In this variation, we can also assign which hotels are allowed by each company or for each employee. E.g. A company may allow bookings for standard rooms and junior suites for hotel "1" and but only allow standard room for hotel "2". Same for employee policies.

4. Overbooking policy

Additional rule: some hotels may allow a small percentage of overbooking. E.g. 5%.

5. Strict component boundaries

In this variation, each service represents the public interface of a business component (functional area or bounded context). All the internals should not be accessible by any other service. Components can only interact via their public interface. E.g. Service A can talk to Service B, but not with Repository B.

6. Sequence Diagram Only

In this variation, you don't need to write any tests or code. You should only create a sequence diagram at code level (collaborators, methods, parameters, return types, data structures, etc) representing the detailed solution. The sequence diagram should represent the interaction of the 3 actors with the services and the internals of each service, including their collaboration with other services and internal collaborators.

outside-in TDD design