Java Class: UserBookingService - Explanation & Interview Q&A

Overview:

`UserBookingService` is a service class in a ticket booking system (IRCTC-like), responsible for user-related operations such as login, sign-up, ticket cancellation, fetching bookings, and train search. It interacts with user and train data, stored in JSON files using Jackson's ObjectMapper.

Line-by-Line Explanation

Class Declaration & Imports

```
package ticket.booking.services;
```

- Defines the package location of the class.

```
import com.fasterxml.jackson.core.type.TypeReference;
import com.fasterxml.jackson.databind.ObjectMapper;
```

- Used to handle JSON parsing and mapping to Java objects.

```
import ticket.booking.entities.Train;
import ticket.booking.entities.User;
import ticket.booking.util.UserServiceUtil;
```

- Custom imports for `User`, `Train` entities, and a utility for password hashing/checking.

```
import java.io.*;
import java.util.*;
```

- Standard Java imports for file handling, collections, and input.

Fields

```
private static final String USER_PATH = "app/src/main/java/ticket/booking/
users.json";
```

- Path to the JSON file storing user data.

```
private User user;
private List<User> userList;
public ObjectMapper objectMapper = new ObjectMapper();
```

- `user`: currently logged-in user.
- `userList`: loaded list of all users.
- `objectMapper`: Jackson utility for JSON I/O.

Constructor with User

```
public UserBookingService(User user1) throws IOException {
    this.user = user1;
    File users = new File(USER_PATH);
    userList = objectMapper.readValue(users, new TypeReference<List<User>>()
{});
}
```

- Initializes the object with a user and loads users from file.

Default Constructor (Incomplete)

```
public UserBookingService() {
    File users = new File(USER_PATH);
}
```

- Only creates file reference, **does not load** `userList` - should be fixed.

Login User

```
public Boolean loginUser() {
    Optional<User> foundUser = userList.stream()
        .filter(user1 -> user1.getName().equals(user.getName()) &&
                UserServiceUtil.checkPassword(user.getPassword(),
user1.getHashedPassword()))
        .findFirst();
```

```
    return foundUser.isPresent();
}
```

- Finds matching user by name and password. Returns `true` if found.

Sign Up User

```
public Boolean signUp(User user1) {
    try {
        userList.add(user1);
        saveUserListToFile();
        return Boolean.TRUE;
    } catch (IOException ex) {
        return Boolean.FALSE;
    }
}
```

- Adds a new user to the list and saves it to file.

Save Users to File

```
private void saveUserListToFile() throws IOException {
    File usersFile = new File(USER_PATH);
    objectMapper.writeValue(usersFile, userList);
}
```

- Saves the current user list to the JSON file.

Fetch Booking

```
public void fetchBooking() {
    Optional<User> userFetched = userList.stream()
        .filter(user1 -> user.getName().equalsIgnoreCase(user1.getName()) &&
                UserServiceUtil.checkPassword(user.getPassword(),
user1.getHashedPassword()))
        .findFirst();
    if (userFetched.isPresent()) {
        userFetched.get().printTickets();
    }
}
```

- Finds the logged-in user and prints their tickets.

Cancel Booking

```
public Boolean cancelBooking(String ticketId) {
    Scanner s = new Scanner(System.in);
    System.out.println("Enter the ticket id to cancel");
    ticketId = s.next();

    if (ticketId == null || ticketId.isEmpty()) {
        System.out.println("Ticket ID cannot be null or empty.");
        return Boolean.FALSE;
    }

    boolean removed = user.getTicketsBooked().removeIf(ticket ->
ticket.getTicketId().equals(ticketId));

    if (removed) {
        System.out.println("Ticket with ID " + ticketId + " has been
canceled.");
        return Boolean.TRUE;
    } else {
        System.out.println("No ticket found with ID " + ticketId);
        return Boolean.FALSE;
    }
}
```

- Reads ticket ID from user, removes matching ticket if present.

Search Trains

```
public List<Train> getTrains(String source, String destination) {
    try {
        TrainService trainService = new TrainService();
        return trainService.searchTrains(source, destination);
    } catch (IOException ex) {
        return new ArrayList<>();
    }
}
```

- Calls TrainService to find trains between source and destination.

Fetch Seats

```
public List<List<Integer>> fetchSeats(Train train) {
    return train.getSeats();
}
```

- Returns a 2D list representing the train's seating layout.

Common Interview Questions with Answers

1. What is the role of ObjectMapper?

Answer: ObjectMapper is a Jackson class used to serialize and deserialize Java objects to/from JSON.

2. Why is Optional used in ``?

Answer: To safely handle the possibility that the user might not be found without using `null`.

3. What design pattern is used here?

Answer: Service Layer pattern – encapsulates business logic related to users.

4. What improvements would you make to this class?

Answer:

- Fix default constructor to load users.
- Avoid reading `ticketId` from Scanner when it's already a parameter.
- Use logging instead of `System.out.println()`.

5. How is password checking handled?

Answer: It uses `UserServiceUtil.checkPassword()` to compare the plain password with the stored hashed password.

6. What happens if the JSON file is corrupted or missing?

Answer: An IOException is thrown, which should be handled to prevent application crash.

7. Can this service handle multiple users at once (concurrently)?

Answer: Not safely. `userList` is shared and not thread-safe. You'd need synchronization or a database.

8. How would you store data in production?

Answer: Use a database like PostgreSQL instead of a JSON file. JSON is fine for local testing or prototypes.

9. What would you test in this class?

Answer:

- Login with valid/invalid credentials
- Signup and verify JSON update
- Booking fetch with no tickets
- Cancel valid/invalid ticket

10. What libraries does this class depend on?

Answer:

- Jackson (`ObjectMapper`)
- Java Collections
- Custom classes: `User`, `Train`, `UserServiceUtil`

Would you like a cleaned-up refactored version of this class next?