Week 2 Review - National Park Campsite Part 2

In this exercise, you'll continue working with the National Park Campsite Reservation database. You'll now create DAO methods and integration tests.

Step One: Getting started

Open the project

1. In Eclipse, select **File > Import...**.
2. In the import window, expand **Maven**, select **Existing Maven Projects**, and click **Next >**.
3. Click **Browse...** and select the directory that contains the `pom.xml` file of this project.
4. Click **Finish** to import the project.

Database

In the `/database` folder, there's a `campground.sql` SQL script that drops and recreates the tables and data in the `campground` database. Remember to run this script before you work on the exercises below to ensure that your database is in a good state.

Step Two: Review starting code and database schema

Starting code

The curriculum team created the models, DAO classes, and interfaces for you. Take a moment to look through the classes and tests to familiarize yourself with them.

Integration tests

The integration tests are in the `com.techelevator.dao.jdbc` package in the directory `/src/test/java`.

There's a `BaseDAOTests` class that all the tests inherit from. This base class contains the following functionality:

- Creates a new datasource connection to the `campground` database.
- Loads test data.
- Closes the datasource connection after all tests have completed.
- Rolls back the test data and any changes to the database after the tests have completed.

The requirements below explain what data you'll need to provide for the tests that you write.

Connection strings

Verify that the connection string in `BaseDAOTests` is correct.

Database schema

To refresh your memory, here's the database schema:

Parks table

A parks table is provided to the system that provides the data for each of the supported national parks. The data columns are as follows:

	Field	Description
PK	park_id	A surrogate key for the park.
	name	The name of the park.
	location	The location of the park.
	establish_date	The date that the park was established.
	area	The size of the park in square kilometers.
	visitors	The annual number of visitors to the park.
	description	A short description about the park.

Campground table

A campground table is provided to the system that provides a list of the one or many campgrounds located inside of a national park. The data columns are as follows:

	Field	Description
PK	campground_id	A surrogate key for the campground.
FK	park_id	The park that the campground is associated with.
	name	The name of the campground.
	open_from_mm	The numerical month the campground is open for reservation. (01 - January, 02 - February, ...)
	open_to_mm	The numerical month the campground is closed for reservation. (01 - January, 02 - February, ...)
	daily_fee	The daily fee for booking a campsite at this campground.

Site table

A site table lists all of the available campsites available for reservation in a campground. The data columns are as follows:

	Field	Description
PK	site_id	A surrogate key for the campsite.
FK	campground_id	The campground that the park belongs to.
	site_number	The arbitrary campsite number.
	max_occupancy	Maximum occupancy at the campsite.
	accessible	Indicates whether or not the campsite is handicap accessible.

	Field	Description
	max_rv_length	The maximum RV length that the campsite can fit. 0 indicates that no RV fits at this campsite.
	utilities	Indicates whether or not the campsite provides access to utility hookup.

Reservation table

The reservation table lists all of the past, current, and future reservations for a campsite in the national park system. The data columns are as follows:

	Field	Description
PK	reservation_id	A surrogate key for the reservation.
FK	site_id	The campsite the reservation is for.
	name	The name for the reservation.
	from_date	The start date of the reservation.
	to_date	The end date of the reservation.
	create_date	The date the reservation was booked.

Step Three: Create DAO methods and tests

You'll now create the DAO methods for each of the requirements below. The first four have an empty method for you to complete and an integration test for you to verify your work. The last three require you to write the method and test.

List all parks

The application needs the ability to view a list of the parks in the system, sorted alphabetically by location.

A park includes an ID, name, location, established date, area, annual visitor count, and description.

Class	Method	Test
JDBCParkDAO	getAllParks()	getParksTest_Should_ReturnAllParksInLocationAlphaOrder

List all campgrounds for a park

The application needs the ability to view a list of all campgrounds for a park.

A campground includes an ID, name, open month, closing month, and a daily fee.

Class	Method	Test
JDBCCampgroundDAO	getCampgroundsByParkId(int parkId)	getCampgroundsTest_Should_ReturnAllCampgrounds

List all sites in park that allow RVs

The application needs the ability to list all sites in a park—across all campgrounds—that allow RVs.

A site includes an ID, site number, maximum occupancy, if it's accessible, maximum RV length, and if it has utilities.

A site with `0` listed for `max_rv_length` indicates the site doesn't allow RVs.

Class	Method	Test
JDBCSiteDAO	getSitesThatAllowRVs(int parkId)	getSitesThatAllowRVs_Should_ReturnSites

Book a reservation

The application needs the ability to book a reservation at a selected campsite.

A reservation requires a site ID, name to reserve under, a start date, and an end date.

The user sees a confirmation ID once they submit their reservation.

Class	Method	Test
JDBCReservationDAO	createReservation(int siteId, String name, LocalDate fromDate, LocalDate toDate)	createReservation_Should_ReturnNewReservationId

Upcoming reservations

The application needs the ability to view a list of all upcoming reservations within the next 30 days for a selected park.

A reservation includes a reservation ID, site ID, name, start date, end date, and date created.

Find the correct classes where you'll need to write this method and test. The test data returns two reservations for test `parkId` 99.

Identify currently available sites in a given park

The application needs the ability to search for currently available sites in a given park for a customer who didn't make a reservation.

A site is unavailable if there's a current reservation for the site.

Find the correct classes where you'll need to write this method and test. The test data returns two sites for test `parkId` 99.

Bonus: Identify available sites for a future reservation

The application needs the ability to search for the availability of all sites in a park given a start and end date so that a user can make a reservation.

A reservation search only requires the desired park, a start date, and an end date.

A site is unavailable if any part of the user's preferred date range overlaps with an existing reservation.

Find the correct classes where you'll need to write this method and test.

Test data

If you use the start date as today + 3, and the end date as today + 5, and `parkId` 99, the query returns two sites.

> Tip: Use the `LocalDate.now().plusDays(long daysToAdd)` method to get the dates needed.