Implementing Query Methods in `DBDataService`

In this tutorial, we will implement the following methods in the `DBDataService` class to retrieve flights based on specific criteria:

- `getFlightsByOrigin(String airportCode)`
- `getFlightsByCarrier(String carrierCode)`
- `getFlightsByState(String stateName)`

These methods will utilize Spring Data JPA repository queries to fetch data from the database.

Step 1: Update the `FlightRepository` Interface

First, add custom query methods to the `FlightRepository` interface.

```
package com.luxoft.flights.repositories;

import com.luxoft.flights.model.Flight;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.Date;
import java.util.List;
import java.util.Optional;

@Repository
public interface FlightRepository extends JpaRepository<Flight, Long> {
    Optional<Flight> findByTailNumAndFlightDate(String tailNum, Date flightDate);

    // Add the following methods:
    List<Flight> findByOriginAirportCode(String airportCode);

    List<Flight> findByCarrierCode(String carrierCode);

    List<Flight> findByOriginStateNameOrDestinationStateName(String originStateName,
String destStateName);
}
```

Explanation:

- **`findByOriginAirportCode`**: Fetches flights where the origin airport code matches the given `airportCode`.
- **`findByCarrierCode`**: Retrieves flights operated by the carrier with the specified `carrierCode`.
- **`findByOriginStateName`**: Obtains flights where the origin airport's state name matches the provided `stateName`.

These methods leverage Spring Data JPA's query creation from method names, navigating through entity relationships.

Step 2: Implement Methods in `DBDataService`

Next, implement these methods in the `DBDataService` class by invoking the repository methods.

```
package com.luxoft.flights.services;

import com.luxoft.flights.model.Flight;
import com.luxoft.flights.repositories.FlightRepository;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class DBDataService implements DataService {

    private final FlightRepository flightRepository;

    // Existing methods...

    @Override
    public List<Flight> getFlightsByOrigin(String airportCode) {
        return flightRepository.findByOriginAirportCode(airportCode);
    }

    @Override
    public List<Flight> getFlightsByCarrier(String carrierCode) {
        return flightRepository.findByCarrierCode(carrierCode);
    }

    @Override
    public List<Flight> getFlightsByState(String stateName) {
        return flightRepository.findByOriginStateNameOrDestinationStateName(stateName,
stateName);
    }

}
```

Explanation:

- Each method calls the corresponding repository method to fetch flights based on the specified criterion.

Additional Tips

- **Property Naming:** Ensure the property names in your query methods match the exact names

in your entities. For nested properties, use the dot notation in JPQL or the camel case in method names.

- **Lazy Loading:** Be cautious of `FetchType.LAZY` when accessing related entities to avoid `LazyInitializationException`. You may need to adjust fetch strategies or transactional boundaries.
- **Custom Queries:** If you require more complex queries, consider using the `@Query` annotation with JPQL or native SQL.

Using `@Query` Annotation:

```
@Repository
public interface FlightRepository extends JpaRepository<Flight, Long> {

    @Query("SELECT f FROM Flight f WHERE f.origin.airportCode = :airportCode")
    List<Flight> findFlightsByOriginCode(@Param("airportCode") String airportCode);

    @Query("SELECT f FROM Flight f WHERE f.carrier.code = :carrierCode")
    List<Flight> findFlightsByCarrierCode(@Param("carrierCode") String carrierCode);

    @Query("SELECT f FROM Flight f WHERE f.origin.state.name = :stateName")
    List<Flight> findFlightsByOriginStateName(@Param("stateName") String stateName);
}
```

- **Explanation:** The `@Query` annotation allows you to define JPQL queries when method naming conventions become cumbersome or insufficient.

Conclusion

By following these steps, you have successfully implemented the methods to retrieve flights based on origin airport code, carrier code, and state name using Spring Data JPA repositories. This enhances your application's data retrieval capabilities and leverages the power of JPA for efficient database interactions.