Using @Primary and @Qualifier Annotations

Spring Feature Description

The @Primary annotation indicates that a particular bean should be given preference when multiple candidates are qualified to autowire a single-valued dependency. The @Qualifier annotation allows you to specify which bean to inject when there are multiple candidates.

Why Do We Need It in Flight Application?

In our application, we may have multiple implementations of a service or component (e.g., different data loaders for various formats). By using @Primary and @Qualifier, we can control which bean gets injected without altering our application logic.

Refactoring Application

We will introduce two different data loaders, one for CSV files and another for database interactions, and use these annotations to manage them.

The Updated Code

- **Introducing DataLoader Interface**

Define a common interface for data loaders:

```
public interface DataLoader {
    void loadData() throws Exception;
}
```

- **Define Multiple Implementations**

Define two data loader classes:

```
import org.springframework.stereotype.Component;

@Component
public class CSVDataLoader implements DataLoader {
    // Implementation for loading from CSV
}

@Component
public class DatabaseLoader implements DataLoader {
    // Implementation for loading from database
}
```

- **Injecting Data Loader**

Now let's change the FlightsApplication to use the interface DataLoader instead of CSVDataLoader:

```
    @Bean
    CommandLineRunner run(DataService dataService, DataLoader dataLoader) {
        return args -> {
            dataLoader.loadData();
            // Existing code for menu options...
        };
    }
```

Now, when the application starts, it will use the DataLoader interface to load data from either CSV or a database, depending on the implementation injected.

However, if you run the application now, you will get an error: NoQualifyingBeanException because Spring doesn't know which DataLoader implementation to use.

Let's see how to resolve this issue using @Primary and @Qualifier.

- **Use @Primary to Set Default Loader**

Set one of the loaders as primary:

```
import org.springframework.stereotype.Component;
import org.springframework.beans.factory.annotation.Primary;

@Component
@Primary
public class CSVDataLoader implements DataLoader {
    // Implementation for loading from CSV
}
```

Now, if no @Qualifier is specified, the CSVDataLoader will be injected by default.

- **Specifying the @Qualifier**

If you want to use the DatabaseLoader instead, you can specify it using @Qualifier. We need to add the @Qualifier annotation to the DatabaseLoader class. For the common approach, lets also add the @Qualifier annotation to the CSVDataLoader class:

```
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Component;

@Component
@Qualifier("csvDataLoader")
public class CSVDataLoader implements DataLoader {
    // Implementation for loading from CSV
}
```

```java
@Component
@Qualifier("dbDataLoader")
public class DatabaseLoader implements DataLoader {
    // Implementation for loading from database
}
```

Now you can inject the DatabaseLoader using @Qualifier.

- **Injecting with @Qualifier**

Use @Qualifier when injecting the specific loader you want to use:

```java
@SpringBootApplication
public class FlightsApplication {

    @Bean
    CommandLineRunner run(DataService dataService,
        @Qualifier("dbDataLoader") DataLoader dataLoader)
            throws Exception {
        return args -> {
            dataLoader.loadData();
            // Existing data loading logic
        };
    }
}
```

Benefits of Using @Primary and @Qualifier

- **Fine-Grained Control**: Specify exactly which bean to inject when there are multiple candidates.
- **Maintainability**: Easily swap implementations without changing the code that uses them.

Further Advices

Use @Primary sparingly to avoid confusion. Whenever possible, opt for @Qualifier to make dependencies explicit and understandable.