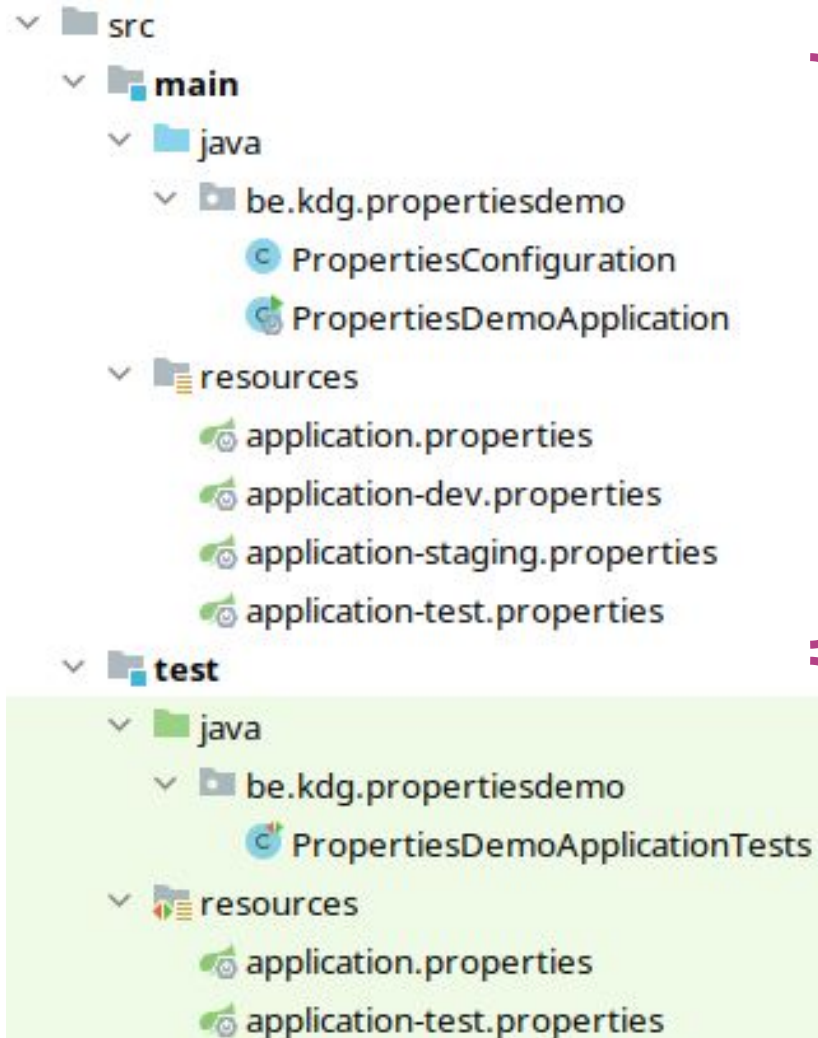


Programming 5

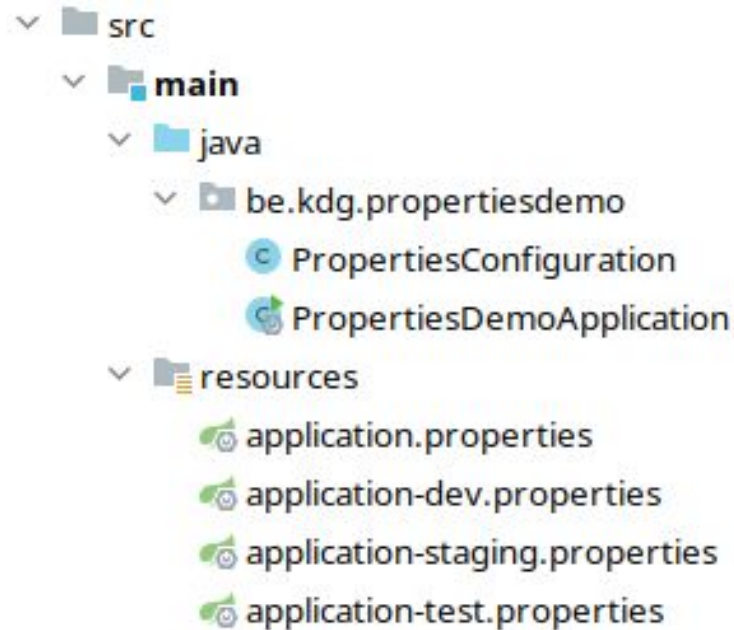
Spring profiles & seeding strategies

Spring Profiles



Two separate
source sets

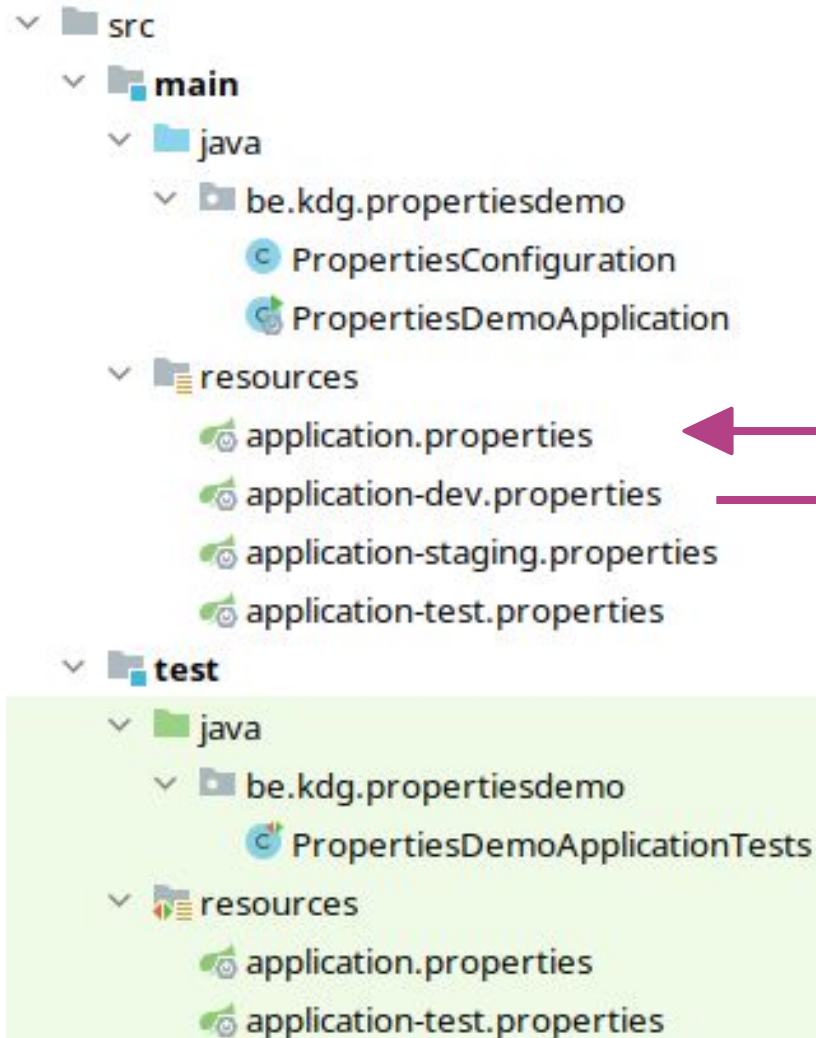
Spring Profiles



Four profiles,
including the default

Two profiles,
including the default

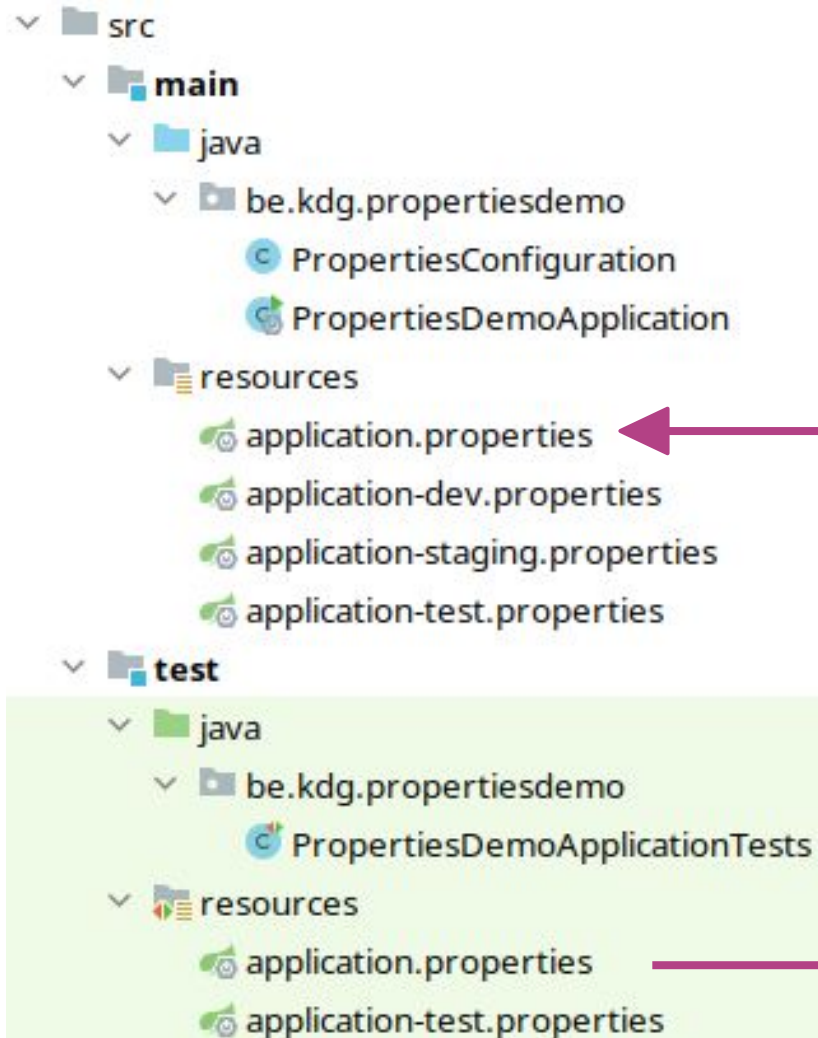
Spring Profiles



When the **dev** profile is active, these are “merged”:

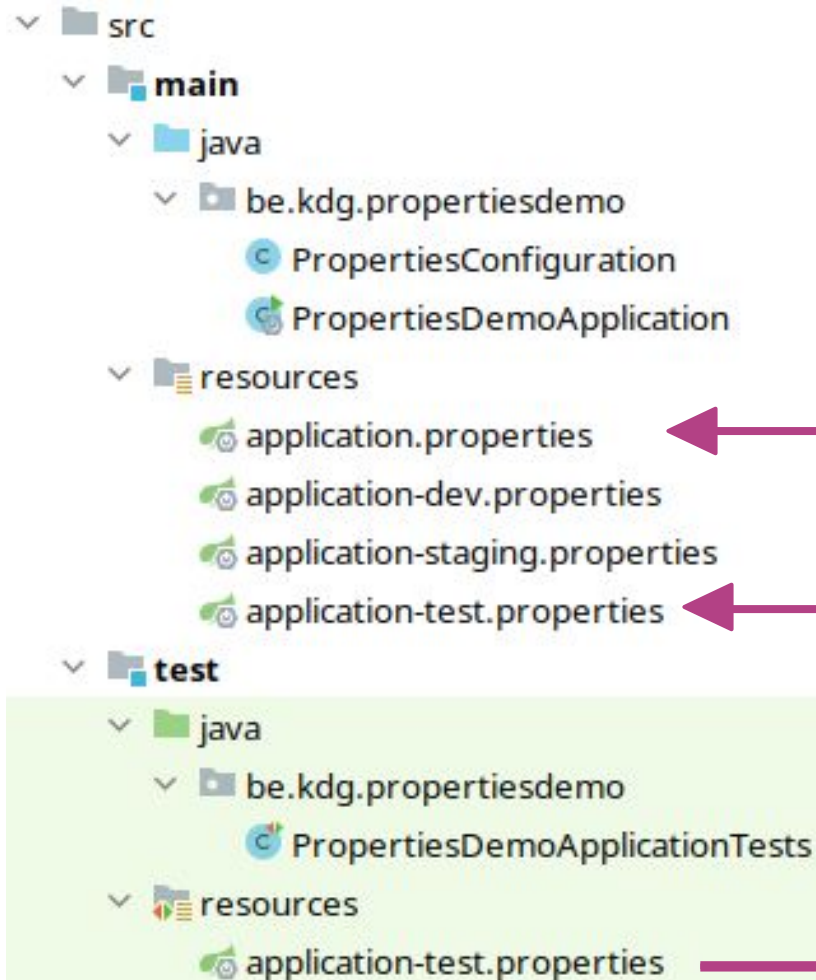
- Properties found in **application-dev.properties** take precedence
- Properties *not* found in **application-dev.properties** will be taken from **application.properties**

Spring Profiles



While executing tests, this one **fully replaces** the other (not merged)

Spring Profiles



Note:

application.properties (test) was removed for the example on this slide!

Combining source sets and profiles.
When executing tests with the **test** profile active:

application-test.properties (main) is fully replaced (= not loaded).

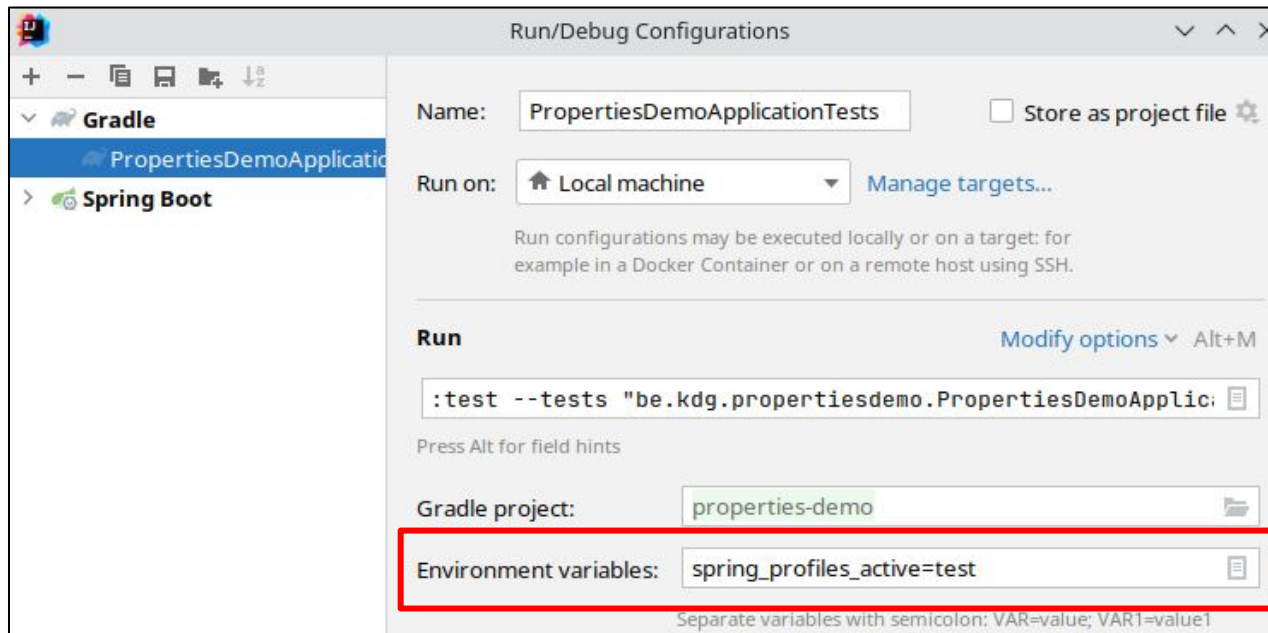
What actually happens:

application-test.properties (test) is merged with
application.properties (main)

Spring Profiles



- Activating profiles for tests:



- Or: `@ActiveProfiles("dev")`

Baeldung



Seeding Strategies

- During tests, you can enable seeding. However, **don't** modify (update/delete) seeded data in your tests, because:
 - There's no guarantee about the order in which tests are executed
 - Tests require a consistent predefined state
 - `@Transactional` tests can cause side-effects
 - Seeding users allows you to easily test with security enabled (`@WithUserDetails`)
- Use `BeforeEach` / `BeforeAll` / `AfterEach` / `AfterAll` to add actual application data for use in tests
- Use an "Arrange" step to set things up (and clean up afterwards)

Seeding Strategies


- **Strategy 1**: a test-specific SQL seed file

test/resources/sql/data.sql

```
INSERT INTO /* ... etc. */
```

test/resources/application.properties

```
spring.sql.init.mode=always  
spring.sql.init.data-locations=classpath:sql/data.sql  
spring.jpa.defer-datasource-initialization=true
```



If you have **sql/data.sql** in *both* main and **test** sourcesets, then the one from **test** will take precedence during the execution of tests.

Seeding Strategies

- **Strategy 2**: using @BeforeAll and/or Arrange

test/java/.../...Test.java

```
@Autowired
private StationRepository stationRepository;

@BeforeAll
public void setup() {
    stationRepository.save(new Station("MAN", "Some description", null));
    stationRepository.save(new Station("XYZ", "Some mandatory descr.", null));
    stationRepository.save(new Station("man", "Some description", null));
    stationRepository.save(new Station("ABC", "Some description", null));
}
```

test/resources/application.properties

```
spring.sql.init.mode=never
```

Seeding Strategies

- **Strategy 3**: using a seeding bean

test/java/.../Seeder.java

```
@Component
@Profile("strategy3")
public class Seeder {
    private final StationRepository stationRepository;

    public Seeder(StationRepository stationRepository) {
        this.stationRepository = stationRepository;
        seed();
    }

    private void seed() {
        stationRepository.save( // ... etc.
    }
}
```

test/resources/application.properties

```
spring.sql.init.mode=never
```

Seeding Strategies

- Out of scope for Programming 5 / Extra information:
 - Don't Use @Transactional in Tests
 - Performance: A new **ApplicationContext** is created for each Spring profile and for each test class with a **@MockBean**

Optimizing Spring Integration

