Spring Boot Techniques

1. Setting app properties at the command line
2. Specifying which properties file to use
3. Defining YAML properties files
4. Using Spring profiles

1. Setting App Properties at the Command Line

- Recap of application properties
- Source of external configuration
- Setting properties at the command line

Demo package: `demo.commandlineproperties`

Recap of Application Properties

- A Spring Boot application can define properties in an `application.properties` file:

```
name=John Smith
```

application.properties

- You can inject properties via `@Value("${propName}")`

```
@Component
public class MyBean1 {

    @Value("${name}")
    private String name;
    …
}
```

MyBean1.java

Source of External Configuration

- Spring Boot lets you define application properties in many places, such as:
  - Command-line arguments
  - Environment variable `SPRING_APPLICATION_JSON`
  - Operating system environment variables
  - Application properties outside your JAR
  - Application properties inside your JAR

Setting Properties at the Command Line

- If you define command-line args that start with `--`
  - Spring Boot converts them into application properties

- E.g., set the `name` property via a command-line arg:

```
--name="Mary Jones"
```

- Let's see an example in IntelliJ…

Pearson

2. Specifying which Properties File to Use

- Location of properties files
- Specifying a different properties file

Pearson

Demo package: `demo.differentpropertiesfile`

Location of Properties Files

- `SpringApplication` looks in the following places to find properties files (highest priority first):
    - `/config` subdirectory of your Java app directory
    - Your Java app directory
    - `/config` package on classpath
    - Root package on classpath

Specifying a Different Properties File (1 of 2)

- You can tell Spring to use a different properties file:

```
@SpringBootApplication
public class Application {

    private static void demo2(String[] args) {

        System.setProperty("spring.config.name", "app2");
        ApplicationContext ctx = SpringApplication.run(Application.class, args);

        …
```

Application.java

```
name=Bill Jones
```

app2.properties

- Alternatively, you can set the `SPRING_CONFIG_NAME` environment variable

Specifying a Different Properties File (2 of 2)

- You can also use a command-line argument to specify which application properties file to use:

```
--spring.config.name=app2
```

- This enables you to specify a properties file as part of your overall CI/CD process
  - E.g. in a Jenkins build script

3. Defining YAML Properties Files

- Overview of YAML files
- Using YAML properties in beans - technique 1
- Using YAML properties in beans - technique 2

Demo package: demo.yamlpropertiesfiles

Overview of YAML Files

- Spring Boot supports YAML as an alternative format for defining application properties:

```
contact:
    tel: 555-111-2222
    email: contact@mydomain.com
    web: http://mydomain.com
```

app3.yml

- YAML is convenient for specifying hierarchical config data

Using YAML Properties in Beans - Technique 1

- Here's one way to use YAML properties in a bean:

```
@Component
public class MyBean3a {

    @Value("${contact.tel}")
    private String tel;

    @Value("${contact.email}")
    private String email;

    @Value("${contact.web}")
    private String web;
    …
}
```

MyBean3a.java

Using YAML Properties in Beans - Technique

- Here's another way to use YAML properties in a bean:

```
@Component
@ConfigurationProperties(prefix="contact")
public class MyBean3b {

    private String tel;
    private String email;
    private String web;
    …
    // Plus getters and setters - these are essential!
 }
```

MyBean3b.java

- You also need this dependency:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-configuration-processor</artifactId>
</dependency>
```

pom.xml

Pearson

4. Using Spring Profiles

- Overview
- Defining profile-specific components
- Defining profile-specific properties
- Setting the active profile

Demo package: `demo.profiles`

Overview

- Spring profiles provide a way to segregate parts of your application configuration
  - So, configuration is only available in certain environments

- For example:
  - "development" profile
  - "production" profile

Defining Profile-Specific Components

- You can annotate component classes with `@Profile`:

```
public interface MyBean4 {}
```

```
@Component
@Profile("development")
public class MyBean4Dev implements MyBean4 {

    @Override
    public String toString() { return "Hello from MyBean4Dev"; }
}
```

MyBean4Dev.java

```
@Component
@Profile("production")
public class MyBean4Prod implements MyBean4 {

    @Override
    public String toString() { return "Hello from MyBean4Prod"; }
}
```

MyBean4Prod.java

Defining Profile-Specific Properties

- You can also define profile-specific properties:

```yaml
apiserver:
  address: 192.168.1.100
  port: 8080
---
spring:
  config:
    activate:
      on-profile: development
apiserver:
  address: 127.0.0.1
---
spring:
  config:
    activate:
      on-profile: production
apiserver:
  address: 192.168.1.120
```

Default values for properties

Properties for "development" profile

Properties for "production" profile

app4.yml

Pearson

Setting the Active Profile

- You must tell Spring what is the active profile
  - Set the `spring.profiles.active` property

- To set the active profile via application properties:

```
spring.profiles.active=development                                app4.properties
```

- To set it at the command-line:

```
--spring.profiles.active=production
```

Summary

- Setting app properties at the command line
- Specifying which properties file to use
- Defining YAML properties files
- Using Spring profiles

Exercise

- Use profiles to define geography-specific properties:

Property	Value if "UK" profile	Value if "US" profile
txfmt.currency	GBP	USD
txfmt.dtformat	dd-MM-yyyy HH:mm:ss	MM-dd-yyyy HH:mm:ss

- Inject these values into a component class named `FinancialTransactionLogger`
  - Implement a `log()` method to output a formatted currency and timestamp - to format the timestamp, use `DateTimeFormatter.ofPattern(dtformat)`

- Set `spring.profiles.active` (hint, you can set comma-separated profiles)

Pearson