Injection Techniques

1. Using Spring Expression Language (SpEL)
2. Working with command-line arguments

Demo project: **`demo-injection-techniques`**

1. Using Spring Expression Language (SpEL)

- Overview of SpEL
- Simple SpEL example
- SpEL scalar expressions
- Using SpEL for collections
- Using SpEL for parameters

Demo package: `demo.spel`

Overview of SpEL

- Spring Expression Language (SpEL) is a Java-like syntax that you can use in various places in Spring:
  - In beans, via `@Value` annotations on fields
  - On parameters in autowired methods
  - Within XML configuration files
  - Etc.

Simple SpEL Example

- Here's a simple SpEL example in a Spring bean:

```
@Component
public class SpelBean {

    @Value("#{ 5 * 7.5 }")
    private double workingWeek;
    …
}
```

SpelBean.java

- Literals you can use in SpEL:
  - Strings enclosed in single quotes
  - Dates, numbers, booleans
  - null

SpEL Scalar Expressions

- You can create an object in a SpEL expression:

```
@Value("#{ new java.util.Date() }")
private Date timestamp;                                    SpelBean.java
```

- You can call a static method, using `T` to denote a type:

```
@Value("#{ T(java.lang.Math).random() * 100.0 }")
private int luckyNumber;                                   SpelBean.java
```

Using SpEL for Collections (1 of 2)

- SpEL can access items in arrays, collections, and maps:

```
@Value("#{ info.cities[9] }")
private String city;

@Value("#{ info.currencies['UK'] }")
private String currency;
```

SpelBean.java

```
@Component
public class Info {
    public List<String> cities() {…}
    public Map<String, String> currencies() {…}
}
```

Info.java

Using SpEL for Collections (2 of 2)

- SpEL has operators for processing collection items:

```
@Value("#{ info.cities.?[startsWith('B')] }")
private List<String> allBCities;

@Value("#{ info.cities.^[startsWith('B')] }")
private String firstBCity;

@Value("#{ info.cities.$[startsWith('B')] }")
private String lastBCity;

@Value("#{ info.cities.![toUpperCase()] }")
private List<String> upperCities;
```

SpelBean.java

Using SpEL for Parameters

- You can use SpEL for autowired method parameters:

```
@Component
public class SpelBean {

    @Autowired
    public void setUserName(@Value("#{systemProperties['user.name'] }") String n) {
        …
    }
```

SpelBean.java

2. Working with Command-Line Arguments

- Overview of command-line arguments
- Accessing command-line arguments
- Two types of command-line arguments
- Passing command-line arguments
- Accessing command-line arguments

Demo package: `demo.commandlineargs`

Overview of Command-Line Arguments

- Here's a reminder of how to "run" a Spring Boot app:

```
public static void main(String[] args) {

    ApplicationContext ctx = SpringApplication.run(Application.class, args);
    …
}
```

- Note we've passed `args` into `SpringApplication.run()`
  - This makes the command-line args available to your components

Accessing Command-Line Arguments

- You can autowire command-line args into a component:

```
@Component
public class MyBeanWithArgs {

    @Autowired
    public MyBeanWithArgs(ApplicationArguments args) {
        // You can access command-line arguments here…
    }
    …
}
```

MyBeanWithArgs.java

Two Types of Command-Line Arguments

- Spring Boot supports two types of command-line arguments:
- Option arguments, prefixed by ––

```
--target=windows --target=macOS --db=h2
```

- Non-option arguments

```
norway oslo krone 42
```

Passing Command-Line Arguments

- To pass command-line arguments using IntelliJ:
  - Click Run | Edit Configurations
  - Choose the project and class to run
  - Enter program arguments
  - Then run the configuration

Accessing Command-Line Arguments

- The `ApplicationArguments` class has various methods for accessing command-line arguments:
  - `getSourceArgs()`
  - `getOptionNames()`
  - `getOptionValues(optionName)`
  - `getNonOptionArgs()`

- Example:
  - See `MyBeanWithArgs.java`

Summary

- Using Spring Expression Language (SpEL)
- Working with command-line arguments