Spring Framework
Spring Expression Language (SpEL)

TRAINING

LXFT LISTED NYSE

Introduction to Spring Expression Language

- The Spring Expression Language (SpEL) is a powerful expression language that supports querying and manipulating an object graph at runtime.
- SpEL expressions can be used with XML or annotation-based configuration metadata for defining BeanDefinitions.
- In both cases the syntax to define the expression is of the form:

```
#{<expression string>}.
```

Spring Expression Language – Maven Configuration

```xml
<dependencies>
  <dependency>
     <groupId>org.springframework</groupId>
     <artifactId>spring-context</artifactId>
     <version>${spring.version}</version>
  </dependency>
</dependencies>
```

Bean reference using XML-based SpringEL

```
<bean id="personBean" class="UsualPerson">
    <property name="id" value="1"/>
    <property name="name" value="Ivan Ivanov"/>
    <property name="age" value="35"/>
    <property name="country" value="#{countryBean}"/>
    <property name="countryName" value="#{countryBean.name}"/>
    …
</bean>

<bean id="countryBean" class="Country">
    <property name="id" value="1"/>
    <property name="name" value="Russia"/>
    <property name="codeName" value="RU"/>
</bean>
```

ex.1

Bean reference using SpringEL annotation-based configuration

```
@Component("personBean")
public class UsualPerson implements Person {
    @Value("1")
    private int id;

    @Value("Ivan Ivanov")
    private String name;

    @Value("#{countryBean}")
    private Country country;

    @Value("#{countryBean.name}")
    private String countryName;
    …

<context:component-scan base-package="com.luxoft.springel.example02" />
```

ex.2

Task

- Add a `Language` class that contains 3 fields: `id`, `code`, `name`. Inject the language itself and the name into the `Country` class using both XML-based configuration and annotation-based configuration. Extend the unit tests to verify the correct injection of the bean and bean property.

Method invocation with XML-based SpringEL

```
<bean id="personBean" class="UsualPerson">
    <property name="id" value="1"/>
    <property name="name" value="Ivan Ivanov"/>
    <property name="country" value="#{countryBean}"/>
    <property name="fullCountryInfo" value="#{countryBean.toString()}"/>
    …
</bean>

<bean id="countryBean" class="Country">
   <property name="id" value="1"/>
   <property name="name" value="Russia"/>
   <property name="codeName" value="RU"/>
</bean>
```

ex.3

Method invocation with Annotation-based SpringEL

```java
@Component("personBean")
public class UsualPerson implements Person {
    @Value("1")
    private int id;
    @Value("Ivan Ivanov")
    private String name;
    @Value("#{countryBean}")
    private Country country;
    @Value("#{countryBean.toString()}")
    private String fullCountryInfo;
    …
```

```xml
<context:component-scan base-package="com.luxoft.springel.example04" />
```

ex.4

Task

- Add 2 fields `surface` and `population` to the `Country` class. Create a method called `getFullCountryInfo` that will return a `String` to provide full information about the country. Inject this method into the `UsualPerson` class using both XML-based configuration and annotation-based configuration. Extend the unit tests to verify the correct method execution.

Operators with SpringEL – XML-based Configuration

```
public class Numbers {
    private int a;
    private int b;
    …

public class Operators {
   private boolean equalTest;
   private boolean notEqualTest;
   private boolean lessThanTest;
   private boolean lessThanOrEqualTest;
   private boolean greaterThanTest;
   private boolean greaterThanOrEqualTest;
   …
```

Operators with SpringEL – XML-based Configuration

```
<bean id="numbersBean" class="Numbers">
   <property name="a" value="100" />
   <property name="b" value="150" />
     …
</bean>


<bean id="operatorsBean" class="Operators">
   <property name="equalTest" value="#{numbersBean.a == 100}" />
   <property name="notEqualTest" value="#{numbersBean.a != numbersBean.b}" />
   <property name="lessThanTest" value="#{numbersBean.b lt numbersBean.a}" />
   <property name="lessThanOrEqualTest" value="#{numbersBean.c le numbersBean.b}" />
   <property name="greaterThanTest" value="#{numbersBean.d > numbersBean.e}" />
   <property name="greaterThanOrEqualTest" value="#{numbersBean.d >= numbersBean.c}" />
   …
</bean>
```

ex.5

Operators with SpringEL – Annotation-based Configuration

```
@Component("numbersBean")
public class Numbers {

  @Value("100")
  private int a;
  @Value("150")
  private int b;
  …
```

```
<context:component-scan base-package="com.luxoft.springel.example06" />
```

Operators with SpringEL – Annotation-based Configuration

```
@Component("operatorsBean")
public class Operators {
  // Relational operators
  @Value("#{numbersBean.a == 100}") // true
  private boolean equalTest;
  @Value("#{numbersBean.a != numbersBean.b}") // true
  private boolean notEqualTest;
  @Value("#{numbersBean.b < numbersBean.a}") // false
  private boolean lessThanTest;
  @Value("#{numbersBean.c <= numbersBean.b}") // false
  private boolean lessThanOrEqualTest;
  @Value("#{numbersBean.d > numbersBean.e}") // false
  private boolean greaterThanTest;
  @Value("#{numbersBean.d >= numbersBean.c}") // true
  private boolean greaterThanOrEqualTest;
  …
```

ex.6

Regular Expressions with XML-based Spring EL

```
public class Expression {

  private String regEx;
  private String regExResult;
  private String numberResult;
  private String email;

  public Expression() {
      email = "office@luxoft.com";
  }

  …

}
```

Regular Expressions with XML-based Spring EL

```
<bean id="expressionBean" class="Expression">
   <property name="regEx“ value=
   "^[_A-Za-z0-9-\\+]+(\\.[_A-Za-z0-9-]+)@[A-Za-z0-9-]+(\\.[A-Za-z0-
9]+)*(\\.[A-Za-z]{2,})$" />

   <property name="regExResult“ value=
   "#{(expressionBean.email matches '^[_A-Za-z0-9-\+]+(\.[_A-Za-z0-9-]+)*@[A-
Za-z0-9-]+(\.[A-Za-z0-9]+)*(\.[A-Za-z]{2,})$')== true ? '-Yes there is a
match.' : '-No there is no match.' }" />

   <property name="numberResult“ value=
   "#{ ('100' matches '\d+') == true ? '-Yes this is digit.' : '-No this is
not a digit.' }" />
</bean>
```

ex.7

Regular Expressions with Annotation-based Spring EL

```
@Component("expressionBean")
public class Expression {

   @Value("^[_A-Za-z0-9-\\+]+(\\.[_A-Za-z0-9-]+)*
   @[A-Za-z0-9-]+(\\.[A-Za-z0-9]+)*(\\.[A-Za-z]{2,})$")
   private String regEx;

   @Value("#{(expressionBean.email matches expressionBean.regEx)== true ?
   '-Yes there is a match.' : '-No there is no match.' }")
   private String regExResult;

   @Value("#{ ('100' matches '\\d+') == true ? '-Yes this is digit.' :
   '-No this is not a digit.' }")
   private String numberResult;

   <context:component-scan base-package="com.luxoft.springel.example08" />
```

ex.8

Task

- Add a new `url` field to the `Expression` class. Use both XML-based configuration and annotation-based configuration to check the matching with a regular expression. Add a unit test to verify the correct behavior.