**Introduction to SpEL (Spring Expression Language)**

**SpEL** is a powerful expression language that allows you to query and manipulate objects at runtime in a Spring-based application.

It is used in:

* Bean definitions
* Annotations (@Value)
* Conditional logic
* Spring Security expressions
* Spring Data queries

**🛠 Features of SpEL**

| **Feature** | **Description** | **Example** |
| --- | --- | --- |
| **Literal expressions** | Numbers, strings, booleans | 'Spring', 100, true |
| **Arithmetic operators** | +, -, \*, /, % | 10 + 5 → 15 |
| **Relational operators** | ==, !=, <, >, <=, >= | 100 > 50 → true |
| **Logical operators** | and, or, not | true and false → false |
| **Ternary operator** | condition ? true : false | 2 > 1 ? 'yes' : 'no' → 'yes' |
| **Elvis operator (?:)** | Null-safe fallback | name ?: 'default' |
| **Safe navigation (?.)** | Null-safe property access | person?.name |
| **Collection selection** | Filter collection elements | list.?[age > 20] |
| **Collection projection** | Transform collection values | list.![name] |
| **Method invocation** | Invoke methods on objects | 'Hello'.toUpperCase() → HELLO |
| **Bean references** | Refer to Spring beans | @empBean.salary |
| **Object construction** | Create objects | new java.util.Date() |

**🎯 SpEL Expression Evaluation Against a Specific Object**

To evaluate an expression against an object instance:

**✅ Example**

import org.springframework.expression.ExpressionParser;

import org.springframework.expression.spel.standard.SpelExpressionParser;

import org.springframework.expression.spel.support.StandardEvaluationContext;

public class SpELDemo {

public static void main(String[] args) {

Employee emp = new Employee(1, "Manjiri", 50000.0);

ExpressionParser parser = new SpelExpressionParser();

StandardEvaluationContext context = new StandardEvaluationContext(emp);

String name = parser.parseExpression("ename").getValue(context, String.class);

Double bonus = parser.parseExpression("salary \* 0.1").getValue(context, Double.class);

System.out.println("Name: " + name); // Output: Manjiri

System.out.println("Bonus: " + bonus); // Output: 5000.0

}

}

**🧾 SpEL in Bean Definition (XML)**

**🎯 Injecting Literal and Computed Values**

<bean id="empbean" class="com.bean.Employee">

<property name="ename" value="#{'Manjiri'.toUpperCase()}"/>

<property name="salary" value="#{25000 + 5000}"/>

</bean>

**🎯 Using Another Bean’s Property**

<bean id="addrbean" class="com.bean.Address">

<property name="city" value="'Pune'"/>

</bean>

<bean id="empbean" class="com.bean.Employee">

<property name="address" value="#{addrbean}"/>

</bean>

**🎯 Conditional Assignment**

<property name="ename" value="#{systemProperties['user.name'] == 'admin' ? 'Admin User' : 'Guest'}"/>

**🧠 Example: SpEL with @Value Annotation**

@Component

public class MyBean {

@Value("#{2 \* 5}")

private int value; // 10

@Value("#{T(java.lang.Math).random() \* 100}")

private double random;

@Value("#{systemProperties['user.name']}")

private String currentUser;

// getters

}

**🔄 Spring EL Parser Utilities**

ExpressionParser parser = new SpelExpressionParser();

Expression expr = parser.parseExpression("'Hello ' + 'World'");

String result = expr.getValue(String.class); // "Hello World"