**Spring Framework**

Official website: <https://spring.io/>

Spring projects: <https://spring.io/projects>

Programming language: Java

Initial release date: June 2003

Developer(s): VMware

License: Apache License 2.0

Platform: Java EE

Stable release: 6.2.0 / 14 November 2024; 9 days ago

**Features**

**Core technologies:** dependency injection, events, resources, i18n, validation, data binding, type conversion, SpEL, AOP.

**Testing:** mock objects, TestContext framework, Spring MVC Test, WebTestClient.

**Data Access:** transactions, DAO support, JDBC, ORM, Marshalling XML.

**Spring MVC and Spring WebFlux web frameworks**.

**Integration:** remoting, JMS, JCA, JMX, email, tasks, scheduling, cache and observability.

**Languages:** Kotlin, Groovy, dynamic languages.

About Spring:

* Refer to the Spring Framework project itself
* Other Spring projects have been built on top of the Spring Framework
* Means the entire family of projects
* Spring Framework is divided into modules. Applications can choose which modules they need. At the heart are the modules of the core container, including a configuration model and a dependency injection mechanism.
* Provides foundational support for different application architectures, including messaging, transactional data and persistence, and web.
* Includes the Servlet-based Spring MVC web framework and, in parallel, the Spring WebFlux reactive web framework

History

* Response to the complexity of the early [J2EE](https://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition) specifications.
* As of Spring Framework 6.0, Spring has been upgraded to the Jakarta EE 9 level (for example, Servlet 5.0+, JPA 3.0+), based on the jakarta namespace instead of the traditional javax packages.
* Spring Framework 6.0 is fully compatible with Tomcat 10.1, Jetty 11 and Undertow 2.3 as web servers, and also with Hibernate ORM 6.1.
* In the early days of J2EE and Spring, applications were created to be deployed to an application server. Today, with the help of Spring Boot, applications are created in a devops- and cloud-friendly way, with the Servlet container embedded and trivial to change.

**Spring Projects:**

* **Spring Framework**
  + Provides a comprehensive programming and configuration model for **modern Java-based enterprise applications**
  + https://spring.io/projects/spring-framework
* **Spring Boot**
  + Makes it easy to **create stand-alone, production-grade Spring based Applications that you can "just run**".
  + https://spring.io/projects/spring-boot
* **Spring Data**
  + Provides a familiar and consistent programming model **for data access**
  + <https://spring.io/projects/spring-data>
  + [Spring Data Commons](https://github.com/spring-projects/spring-data-commons) - Core Spring concepts underpinning every Spring Data module.
  + [**Spring Data JDBC**](https://spring.io/projects/spring-data-jdbc) - Spring Data repository support for JDBC.
  + [Spring Data R2DBC](https://spring.io/projects/spring-data-r2dbc) - Spring Data repository support for R2DBC.
  + [**Spring Data JPA**](https://spring.io/projects/spring-data-jpa) - Spring Data repository support for JPA.
  + [Spring Data KeyValue](https://github.com/spring-projects/spring-data-keyvalue) - Map based repositories and SPIs to easily build a Spring Data module for key-value stores.
  + [Spring Data LDAP](https://spring.io/projects/spring-data-ldap) - Spring Data repository support for [Spring LDAP](https://github.com/spring-projects/spring-ldap).
  + [**Spring Data MongoDB**](https://spring.io/projects/spring-data-mongodb) - Spring based, object-document support and repositories for MongoDB.
  + [Spring Data Redis](https://spring.io/projects/spring-data-redis) - Easy configuration and access to Redis from Spring applications.
  + [Spring Data REST](https://spring.io/projects/spring-data-rest) - Exports Spring Data repositories as hypermedia-driven RESTful resources.
  + [Spring Data for Apache Cassandra](https://spring.io/projects/spring-data-cassandra) - Easy configuration and access to Apache Cassandra or large scale, highly available, data oriented Spring applications.
  + [Spring Data for Apache Geode](https://spring.io/projects/spring-data-geode) - Easy configuration and access to Apache Geode for highly consistent, low latency, data oriented Spring applications.
* **Spring Cloud**
  + Provides tools for developers to quickly build some of the common patterns in distributed systems (e.g. configuration management, service discovery, circuit breakers, intelligent routing, micro-proxy, control bus, short lived microservices and contract testing).
  + <https://spring.io/projects/spring-cloud>
  + Has many projects inside it
* **Spring Cloud Data Flow**
  + Microservice based Streaming and Batch data processing for Cloud Foundry and Kubernetes.
* **Spring Security**
  + powerful and highly customizable authentication and access-control framework.
* **Spring Authorization Server**
  + Provides implementations of the [OAuth 2.1](https://datatracker.ietf.org/doc/html/draft-ietf-oauth-v2-1-05) and [OpenID Connect 1.0](https://openid.net/specs/openid-connect-core-1_0.html) specifications and other related specifications
* **Spring for GraphQL**
  + Provides support for Spring applications built on [GraphQL Java](https://www.graphql-java.com/).
* **Spring Session**
  + provides an API and implementations for managing a user’s session information.
* **Spring Integration**
  + support the well-known[**Enterprise Integration Patterns**](http://www.eaipatterns.com/)**.**
* **Spring HATEOAS**
  + provides some APIs to ease creating REST representations that follow the HATEOAS principle when working with Spring and especially Spring MVC.
* **Spring Modulith**
  + allows developers to build well-structured Spring Boot applications and guides developers in finding and working with [application modules](https://docs.spring.io/spring-modulith/reference/fundamentals.html#modules) driven by the domain**.**
* **Spring REST Docs**
  + Helps to document RESTful services**.**
* **Spring AI**
* **Spring Batch**
* **Spring CLI**
* **Spring Flo**
* **Spring AMQP**
* **Spring LDAP**
* **Spring Shell**
* **Spring for Apache Kafka**
* **Spring for Apache Pulsar**
* **Spring State machine**
* **Spring Web Flow**
* **Spring Web Services**

**Section Summary**

* [Introduction to the Spring IoC Container and Beans](https://docs.spring.io/spring-framework/reference/core/beans/introduction.html)
* [Container Overview](https://docs.spring.io/spring-framework/reference/core/beans/basics.html)
* [Bean Overview](https://docs.spring.io/spring-framework/reference/core/beans/definition.html)
* [Dependencies](https://docs.spring.io/spring-framework/reference/core/beans/dependencies.html)
* [Bean Scopes](https://docs.spring.io/spring-framework/reference/core/beans/factory-scopes.html)
* [Customizing the Nature of a Bean](https://docs.spring.io/spring-framework/reference/core/beans/factory-nature.html)
* [Bean Definition Inheritance](https://docs.spring.io/spring-framework/reference/core/beans/child-bean-definitions.html)
* [Container Extension Points](https://docs.spring.io/spring-framework/reference/core/beans/factory-extension.html)
* [Annotation-based Container Configuration](https://docs.spring.io/spring-framework/reference/core/beans/annotation-config.html)
* [Classpath Scanning and Managed Components](https://docs.spring.io/spring-framework/reference/core/beans/classpath-scanning.html)
* [Using JSR 330 Standard Annotations](https://docs.spring.io/spring-framework/reference/core/beans/standard-annotations.html)
* [Java-based Container Configuration](https://docs.spring.io/spring-framework/reference/core/beans/java.html)
* [Environment Abstraction](https://docs.spring.io/spring-framework/reference/core/beans/environment.html)
* [Registering a LoadTimeWeaver](https://docs.spring.io/spring-framework/reference/core/beans/context-load-time-weaver.html)
* [Additional Capabilities of the ApplicationContext](https://docs.spring.io/spring-framework/reference/core/beans/context-introduction.html)
* [The BeanFactory API](https://docs.spring.io/spring-framework/reference/core/beans/beanfactory.html)

**Introduction to the Spring IoC Container and Beans**

* Inversion of Control (IoC) - A principle/Concept
* Dependency injection (DI) - A specialized form of IoC, design pattern
* DI – Here, objects define their dependencies (that is, the other objects they work with) only through constructor arguments, arguments to a factory method, or properties that are set on the object instance after it is constructed or returned from a factory method.
* The IoC container then injects those dependencies when it creates the bean.
* This process is fundamentally the inverse (hence the name, Inversion of Control) of the bean itself controlling the instantiation or location of its dependencies by using direct construction of classes or a mechanism such as the Service Locator pattern.
* Base packages for Spring framework
  + org.springframework.beans
  + org.springframework.context
* The [BeanFactory](https://docs.spring.io/spring-framework/docs/6.2.0/javadoc-api/org/springframework/beans/factory/BeanFactory.html) interface - provides an advanced configuration mechanism capable of managing any type of object.
* [ApplicationContext](https://docs.spring.io/spring-framework/docs/6.2.0/javadoc-api/org/springframework/context/ApplicationContext.html) is a sub-interface of BeanFactory. It adds:
  + Easier integration with Spring’s AOP features
  + Message resource handling (for use in internationalization)
  + Event publication
  + Application-layer specific contexts such as the WebApplicationContext for use in web applications.
* BeanFactory provides the configuration framework and basic functionality
* ApplicationContext adds more enterprise-specific functionality.

For more information on using the BeanFactory instead of the ApplicationContext, refer [BeanFactory API](https://docs.spring.io/spring-framework/reference/core/beans/beanfactory.html).

**Bean:**

* An object that is instantiated, assembled, and managed by a Spring IoC container
* The objects that form the backbone of your application and that are managed by the Spring IoC container are called beans

Beans, and the dependencies among them, are reflected in the configuration metadata used by a container.

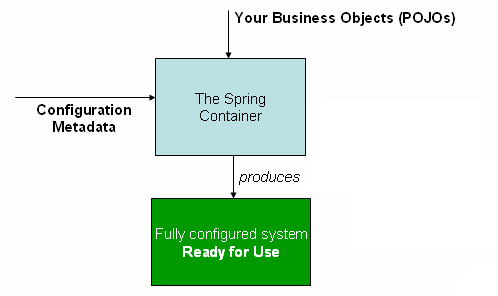
**IoC Container:**

* org.springframework.context.ApplicationContext – Represents IOC container
* Responsible for instantiating, configuring, and assembling the beans
* Implementations of org.springframework.context.ApplicationContext:
  + [AnnotationConfigApplicationContext](https://docs.spring.io/spring-framework/docs/6.2.0/javadoc-api/org/springframework/context/annotation/AnnotationConfigApplicationContext.html)
  + [ClassPathXmlApplicationContext](https://docs.spring.io/spring-framework/docs/6.2.0/javadoc-api/org/springframework/context/support/ClassPathXmlApplicationContext.html)

**Configuration metadata:**

* Provides the instructions needed for the IoC container for instantiating, configuring, and assembling the beans
* Forms
  + Annotated component classes
  + Configuration classes with factory methods
  + External XML files
  + Groovy scripts

**Spring IoC Container - Flow:**



**XML**

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

https://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="..." class="...">

<!-- collaborators and configuration for this bean go here -->

</bean>

<bean id="..." class="...">

<!-- collaborators and configuration for this bean go here -->

</bean>

<!-- more bean definitions go here -->

</beans>

**ApplicationContext context = new ClassPathXmlApplicationContext("services.xml", "daos.xml");**

<beans>

<import resource="services.xml"/>

<import resource="resources/messageSource.xml"/>

<import resource="/resources/themeSource.xml"/>

<bean id="bean1" class="..."/>

<bean id="bean2" class="..."/>

</beans>

DI exists in two major variants:

* [Constructor-based dependency injection](https://docs.spring.io/spring-framework/reference/core/beans/dependencies/factory-collaborators.html#beans-constructor-injection)
* [Setter-based dependency injection](https://docs.spring.io/spring-framework/reference/core/beans/dependencies/factory-collaborators.html#beans-setter-injection)

**Constructor-based Dependency Injection**

<bean id="exampleBean" class="examples.ExampleBean">

<constructor-arg type="int" value="7500000"/>

<constructor-arg type="java.lang.String" value="42"/>

</bean>

**Setter-based Dependency Injection**

<bean id="exampleBean" class="examples.ExampleBean">

<!-- setter injection using the neater ref attribute -->

<property name="integerProperty" value="1"/>

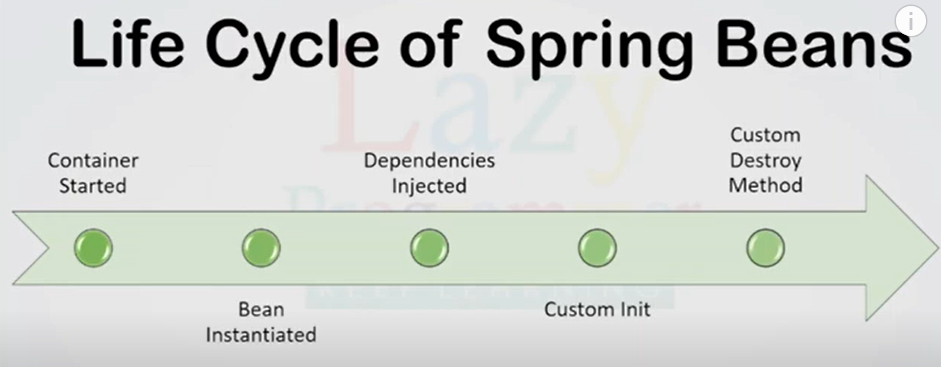
<!-- setter injection using the nested ref element -->

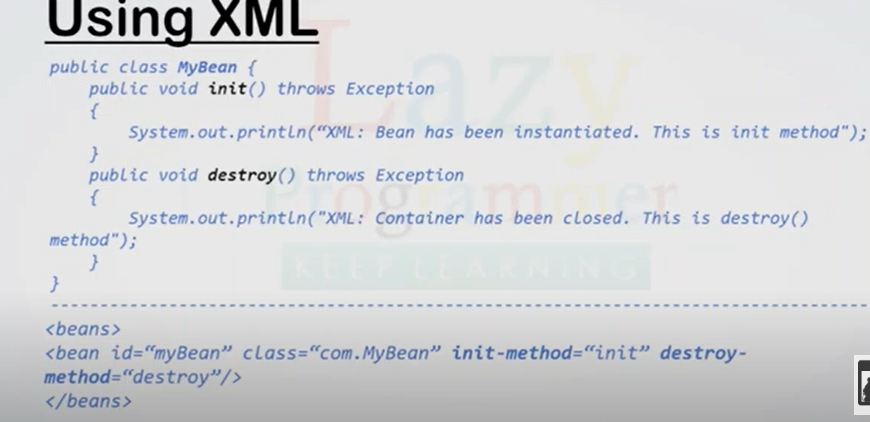
<property name="beanOne">

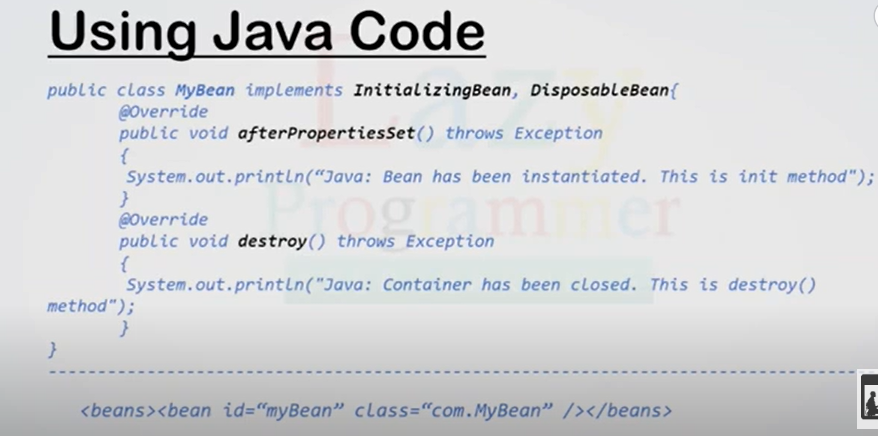
<ref bean="anotherExampleBean"/>

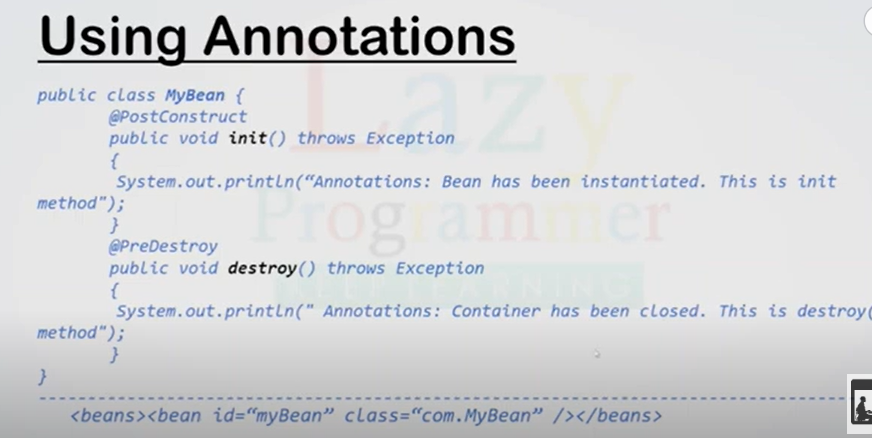
</property>

</bean>









Hook Methods

* Init()
* Destroy()

Options

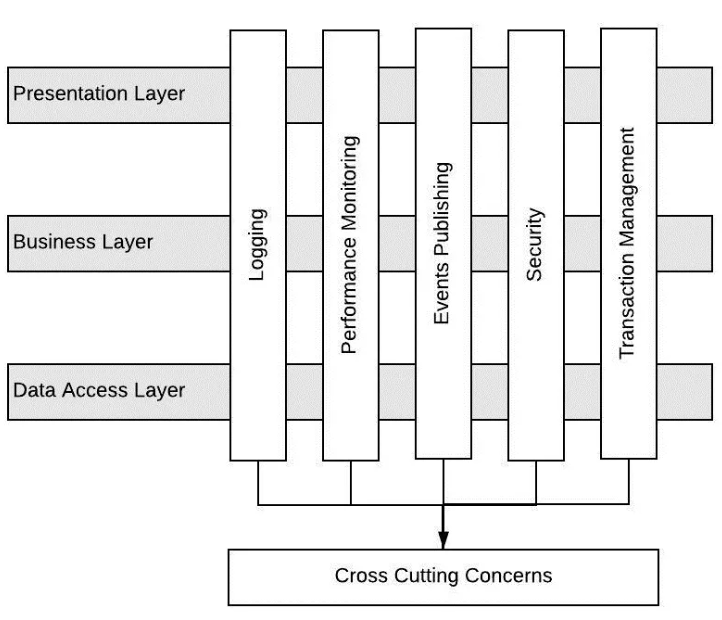
* XML configuration <bean init-method =”” destroy-mothod=””/>
* Java Configuration file- Initialising Bean (afterProeprtiesSet), Disposable bean (destroy())
* Annotations - @PostConstruct(init), @PreDestroy(destroy)

BeanFactory

ApplicationContext ac = new ClassPathXmlApplicationContext(“file.xml”);

**Aspect Oriented Programming and AOP in Spring Framework**

* a programming paradigm that aims to increase modularity by allowing the separation of cross-cutting concerns.
* Cross-cutting concerns are aspects of a program that affect multiple parts of the application, such as logging, security, or transaction management.
* In AOP, these cross-cutting concerns are modularized into separate units called **aspects**.



**Understanding AOP Concepts**

* **Aspect**: The key unit of modularity in AOP, representing a concern that cuts across multiple classes. For example, an aspect for logging can be applied across various methods in different classes.
* **Advice**: This is the action taken by an aspect at a particular join point. There are five types of advice:
  + **Before**: Executed before the method call.
  + **After**: Executed after the method call, regardless of its outcome.
  + **AfterReturning**: Executed after the method returns a result, but not if an exception occurs.
  + **Around**: Surrounds the method execution, allowing you to control the method execution and its result.
  + **AfterThrowing**: Executed if the method throws an exception.
* **Join Point**: A specific point in the execution of a program, such as method execution or exception handling, where an aspect can be applied.
* **Pointcut**: A predicate that matches join points. A pointcut expression specifies where an advice should be applied.
* **Weaving**: The process of linking aspects with the target object. Weaving can occur at compile-time, load-time, or runtime. Spring AOP performs runtime weaving using proxy-based mechanisms.

**Dominant AOP Frameworks**

* **AspectJ**: A powerful and mature AOP framework that supports compile-time and load-time weaving. It offers full AOP support with its own syntax and tools.
* **JBoss AOP**: Part of the JBoss application server, offering integration with Java EE applications.
* **Spring AOP**: A simpler, proxy-based framework that integrates with the Spring Framework, using XML configurations or annotations to define aspects and pointcuts.

<aop:aspectj-autoproxy/>

<bean name="employeeService" class="com.journaldev.spring.service.EmployeeService">

<property name="employee" ref="employee"></property>

</bean>

<!-- Configure Aspect Beans, without this Aspects advices wont execute -->

<bean name="employeeAspect" class="com.journaldev.spring.aspect.EmployeeAspect" />

<bean name="employeeAspectPointcut" class="com.journaldev.spring.aspect.EmployeeAspectPointcut" />

<bean name="employeeAspectJoinPoint" class="com.journaldev.spring.aspect.EmployeeAspectJoinPoint" />

<bean name="employeeAfterAspect" class="com.journaldev.spring.aspect.EmployeeAfterAspect" />

<bean name="employeeAroundAspect" class="com.journaldev.spring.aspect.EmployeeAroundAspect" />

<bean name="employeeAnnotationAspect" class="com.journaldev.spring.aspect.EmployeeAnnotationAspect" />

</beans>

@Aspect

public class EmployeeAspect {

@Before("execution(public String getName())")

public void getNameAdvice(){

System.out.println("Executing Advice on getName()");

}

@Before("execution(\* com.journaldev.spring.service.\*.get\*())")

public void getAllAdvice(){

System.out.println("Service method getter called");

}

}

SpEL

Spring Expression Language

<property name=”a” value=24/>

<property name=”name” value=”ABCD”/>

@Value("#{19 + 1}") // 20

private double add;

@Value("#{'String1 ' + 'string2'}") // "String1 string2"

private String addString;

@Value("#{20 - 1}") // @Value("19")

private double subtract;

@Value("#{10 \* 2}") // 20

private double multiply;

@Value("#{36 / 2}") // 19

private double divide;

@Value("#{36 div 2}") // 18, the same as for / operator

private double divideAlphabetic;

@Value("#{37 % 10}") // 7

private double modulo;

@Value("#{37 mod 10}") // 7, the same as for % operator

private double moduloAlphabetic;

@Value("#{2 ^ 9}") // 512

private double powerOf;

@Value("#{(2 + 2) \* 2 + 9}") // 17

private double brackets;

@Value("#{1 == 1}") // true

private boolean equal;

@Value("#{1 eq 1}") // true

private boolean equalAlphabetic;

@Value("#{1 != 1}") // false

private boolean notEqual;

@Value("#{1 ne 1}") // false

private boolean notEqualAlphabetic;

@Value("#{1 < 1}") // false

private boolean lessThan;

@Value("#{1 lt 1}") // false

private boolean lessThanAlphabetic;

@Value("#{1 <= 1}") // true

private boolean lessThanOrEqual;

@Value("#{1 le 1}") // true

private boolean lessThanOrEqualAlphabetic;

@Value("#{1 > 1}") // false

private boolean greaterThan;

@Value("#{1 gt 1}") // false

private boolean greaterThanAlphabetic;

@Value("#{1 >= 1}") // true

private boolean greaterThanOrEqual;

@Value("#{1 ge 1}") // true

private boolean greaterThanOrEqualAlphabetic;

@Value("#{250 > 200 && 200 < 4000}") // true

private boolean and;

@Value("#{250 > 200 and 200 < 4000}") // true

private boolean andAlphabetic;

@Value("#{400 > 300 || 150 < 100}") // true

private boolean or;

@Value("#{400 > 300 or 150 < 100}") // true

private boolean orAlphabetic;

@Value("#{!true}") // false

private boolean not;

@Value("#{not true}") // false

private boolean notAlphabetic;

@Value("#{2 > 1 ? 'a' : 'b'}") // "a"

private String ternary;

@Value("#{'100' matches '\\d+' }") // true

private boolean validNumericStringResult;

@Value("#{'100fghdjf' matches '\\d+' }") // false

private boolean invalidNumericStringResult;

@Value("#{'valid alphabetic string' matches '[a-zA-Z\\s]+' }") // true

private boolean validAlphabeticStringResult;

@Value("#{'invalid alphabetic string #$1' matches '[a-zA-Z\\s]+' }") // false

private boolean invalidAlphabeticStringResult;

@Value("#{someBean.someValue matches '\d+'}") // true if someValue contains only digits

private boolean validNumericValue;

@Component("carPark")

public class CarPark {

private List<Car> cars = new ArrayList<>();

private Map<String, Car> carsByDriver = new HashMap<>();

public CarPark() {

Car model1 = new Car();

model1.setMake("Good company");

model1.setModel("Model1");

model1.setYearOfProduction(1998);

Car model2 = new Car();

model2.setMake("Good company");

model2.setModel("Model2");

model2.setYearOfProduction(2005);

cars.add(model1);

cars.add(model2);

carsByDriver.put("Driver1", model1);

carsByDriver.put("Driver2", model2);

}

//Getters and setters

}

@Value("#{carPark.carsByDriver['Driver1']}") // Model1

private Car driver1Car;

@Value("#{carPark.carsByDriver['Driver2']}") // Model2

private Car driver2Car;

@Value("#{carPark.cars[0]}") // Model1

private Car firstCarInPark;

@Value("#{carPark.cars.size()}") // Model2

private Integer numberOfCarsInPark;