

TECHNOLOGY



Spring

Ecosystem and Core



Learning Objectives

By the end of this lesson, you will be able to:

- 🕒 Define Spring and list its advantages
- 🕒 Discuss Dependency Injection and list its dependencies
- 🕒 Explain the Spring architecture and install Spring on a system
- 🕒 Get familiarized with the Spring application context and BeanFactory



Learning Objectives

By the end of this lesson, you will be able to:

- 🕒 Understand Constructor and Setter Injection
- 🕒 List and describe Spring annotations
- 🕒 Learn about Spring SpEL



A Day in the Life of a Full Stack Developer

You are hired as a Java Developer in an organization and have been asked to create a web or enterprise application using Java. The application must be high-performing, reusable, and code-based.

You decide to make use of the Spring framework.

To do so, you must explore Spring and its architecture, install it on your system, and work with it.



Spring: Overview

Spring

Spring is a powerful Java corporate application development framework.

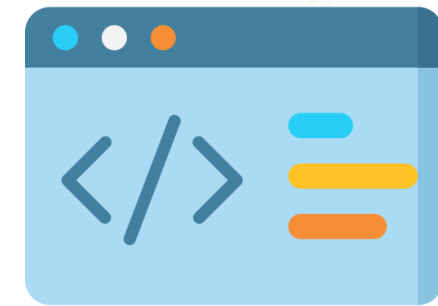
It is challenging to use the Spring framework with:



High performance



Testability

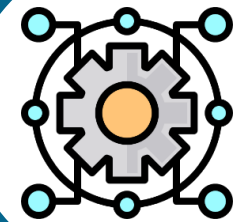


Reusable codes

Spring



Free and open-source Java platform



Distributed under the Apache License 2.0



Light in terms of weight

Spring

The Spring framework can be used to create any type of Java program.

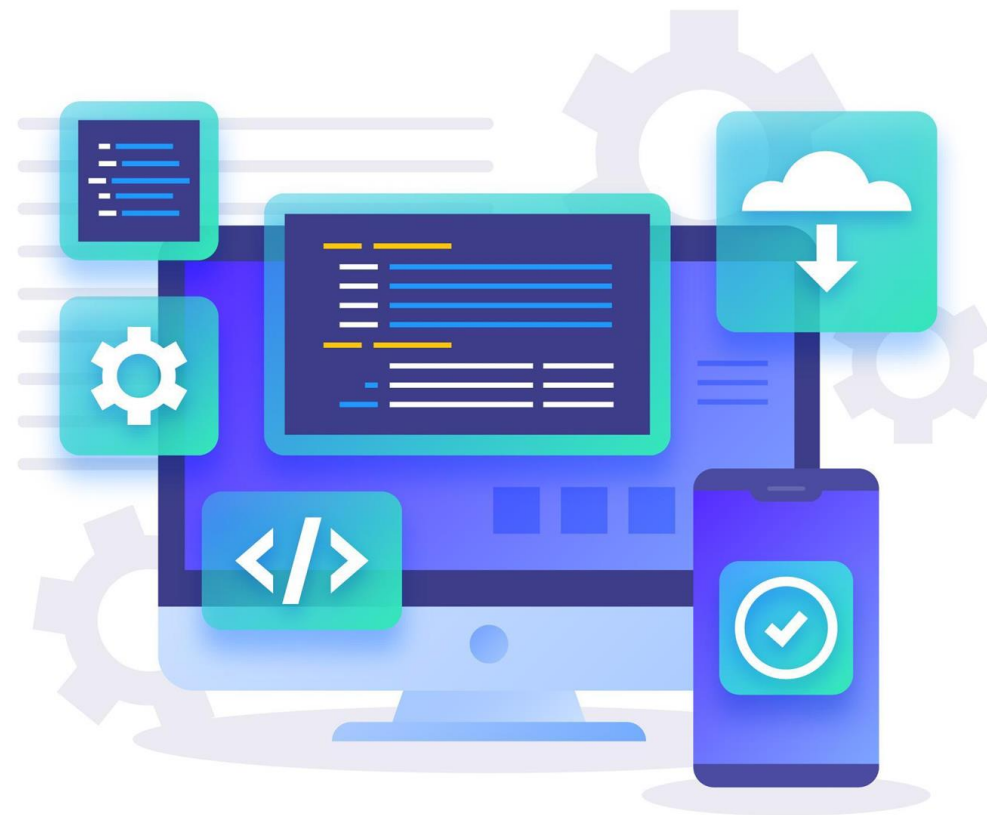


Several extensions can be used to create web applications on top of the Java Enterprise Edition platform.



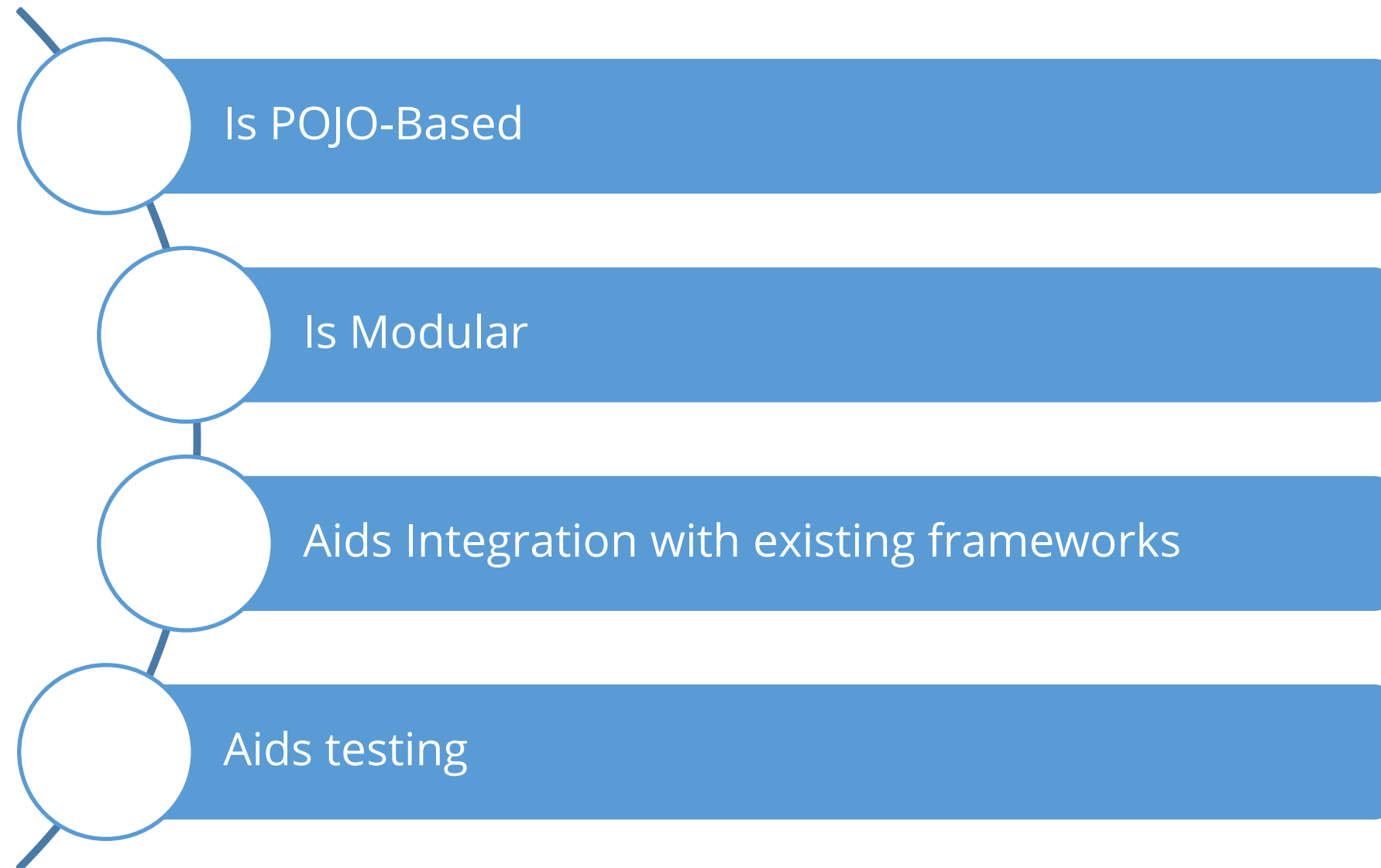
Spring

With the support of the POJO-based programming model, Spring's major goal is to make J2EE development easier to use and provide better programming techniques.



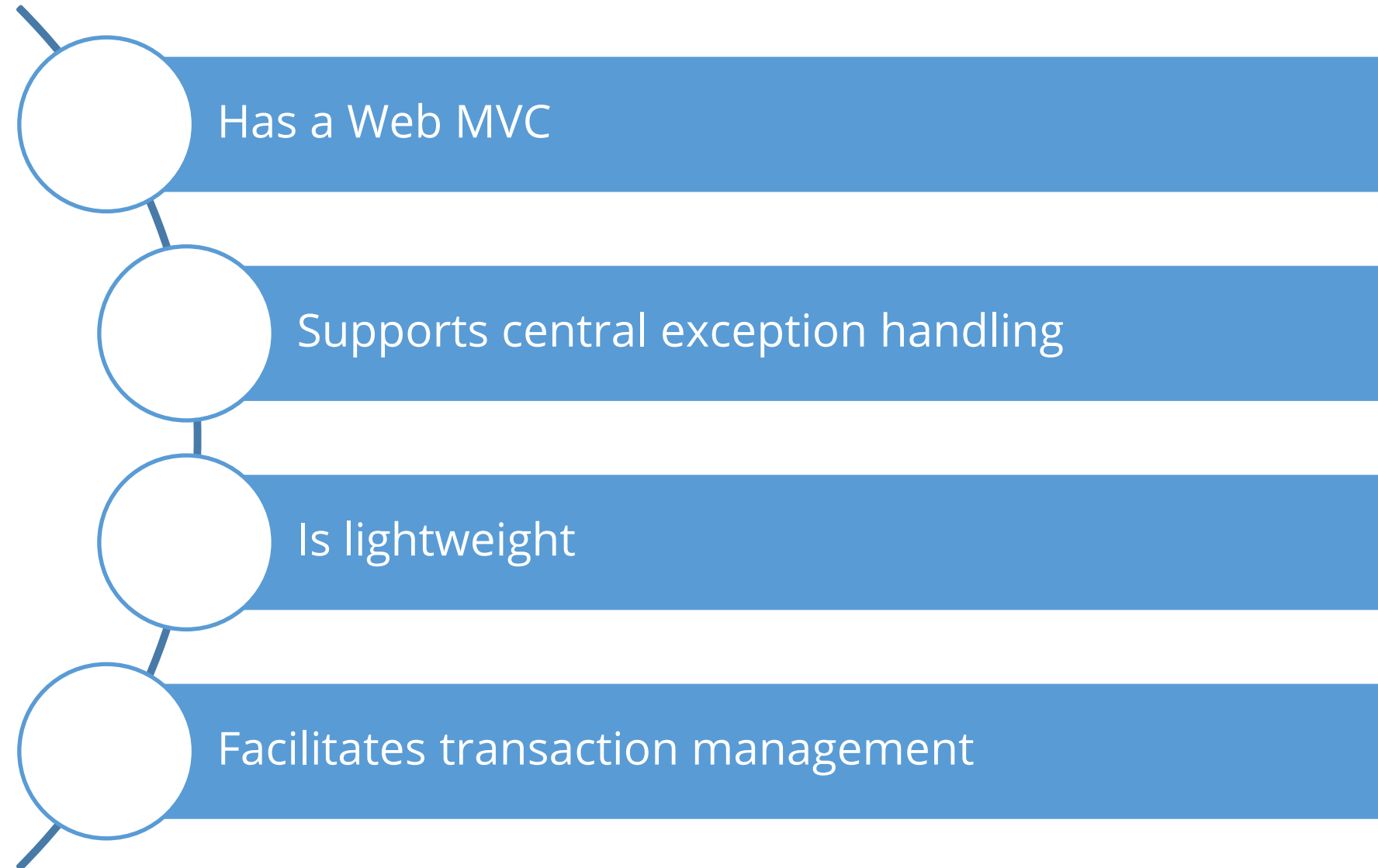
Advantages of Spring

The Spring:



Advantages of Spring

The Spring:



Configuring Spring Core in Java Project



Problem Statement:

You have been asked to configure the Spring Core dependencies in a Java project using Maven and XML configuration.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed are:

1. Creating a Maven project
2. Adding Spring Core dependency
3. Creating a bean class
4. Configuring XML file

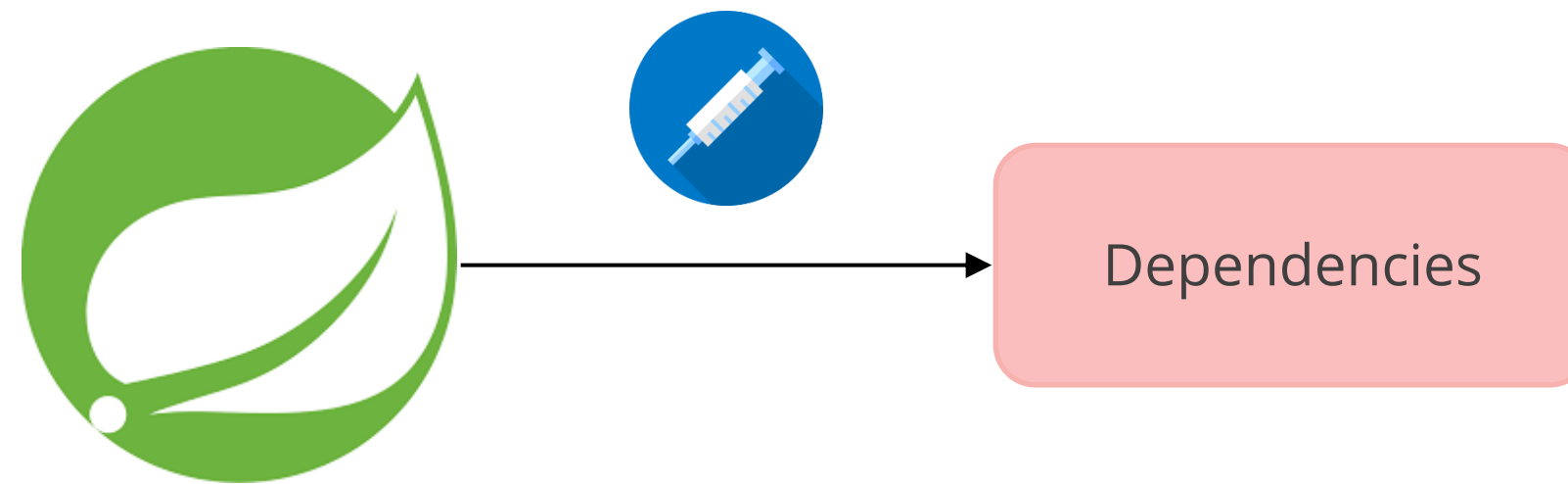


Dependency Injection

Dependency Injection

Dependency Injection is the basic functionality given by Spring IoC.

Spring-Core is responsible for injecting dependencies.



It injects the dependencies with the help of Setter Methods or Constructor



Dependency Injection

The design principles of IoC:

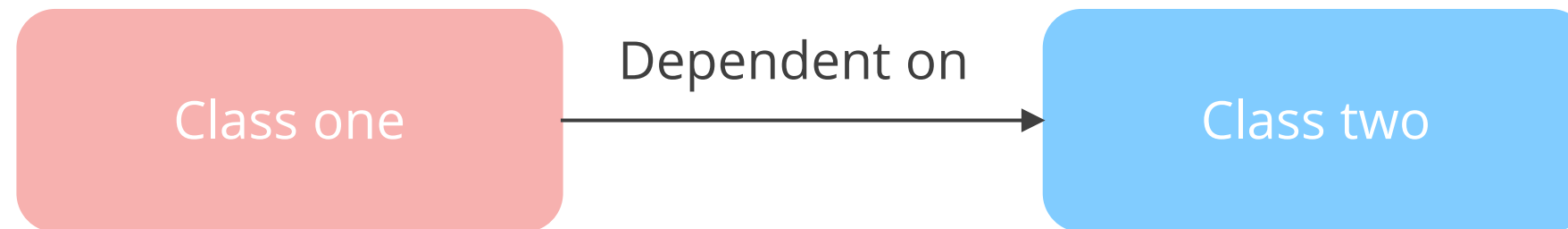
Emphasize keeping the
Java classes
independent from each
other

Get free by container
from creating an
object and
maintenance



Need for Dependency Injection

Dependency causes issues like system failures.



Need for Dependency Injection

Spring IoC uses dependency injection to resolve dependencies by:



Making the code easier to reuse and test



Providing interfaces for shared functionality

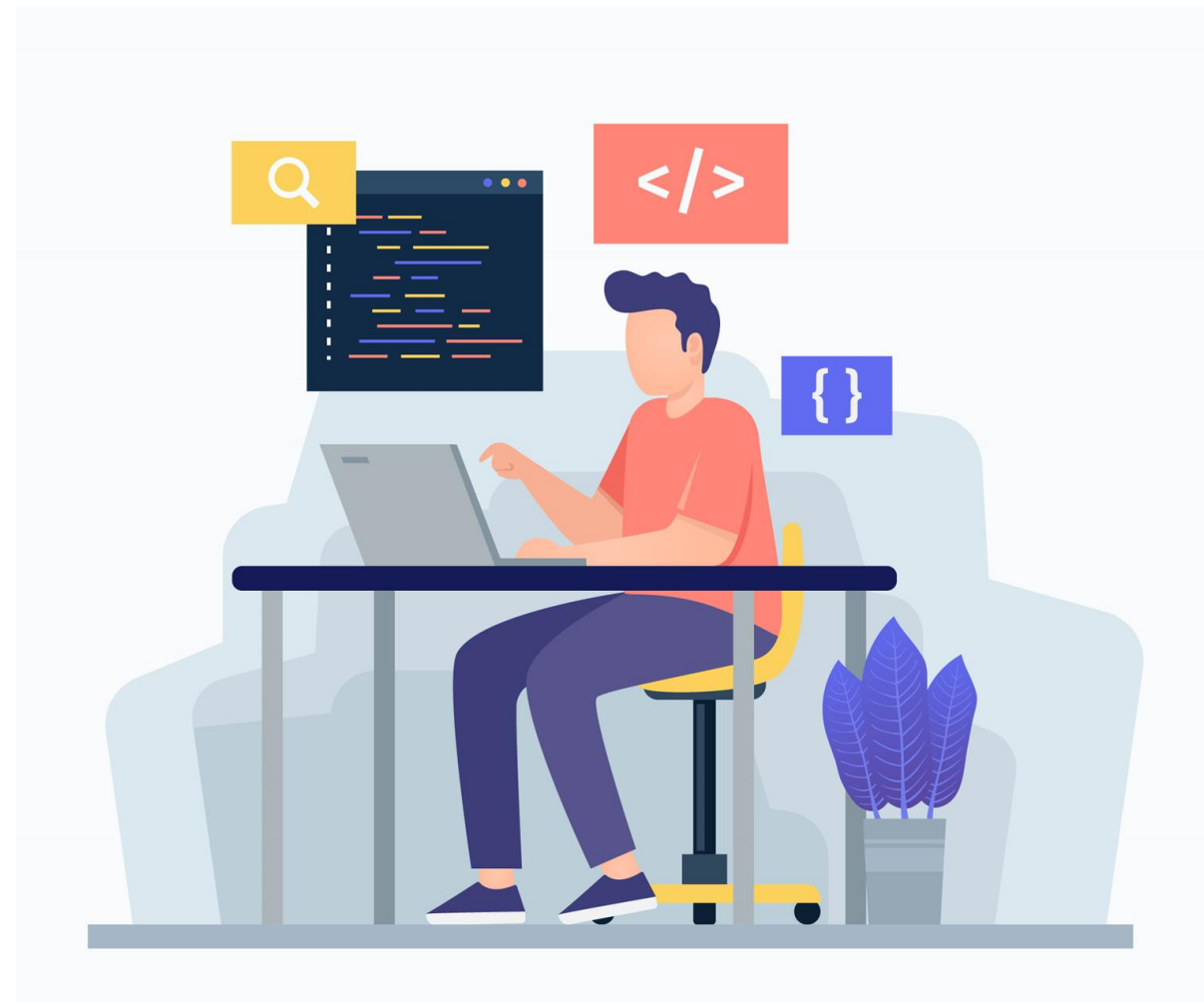


Achieving loose coupling between classes

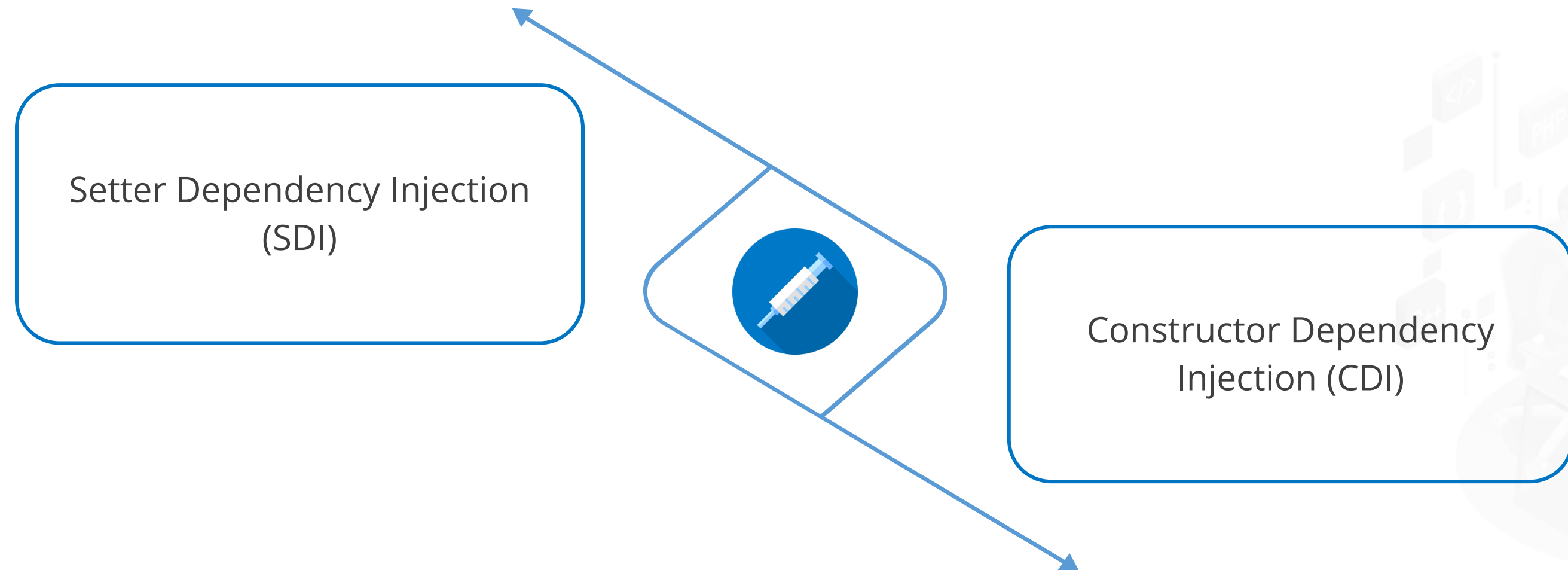


Need for Dependency Injection

The container completes the work of instantiating objects according to the developer's preferences.



Types of Spring Dependency Injection



Setter Dependency Injection

This is the basic Dependency Injection approach. In this case:

1

DI is injected via setter and getter methods.

2

The bean-configuration file is used to configure the DI as SDI in the bean.

3

The property to be set using SDI is stated in the bean-config file's <property> tag.

Constructor Dependency Injection

In case of constructor dependency injection:

1

DI is injected via constructors.

2

The bean-configuration file is used to set the DI as CDI in the bean.

3

The property to be set using the CDI is declared in the bean-config file's constructor-arg tag.

TECHNOLOGY

Spring Architecture

Spring Architecture

Spring may be a one-stop shop for all business apps.

Spring is modular and allows picking and choosing relevant modules.

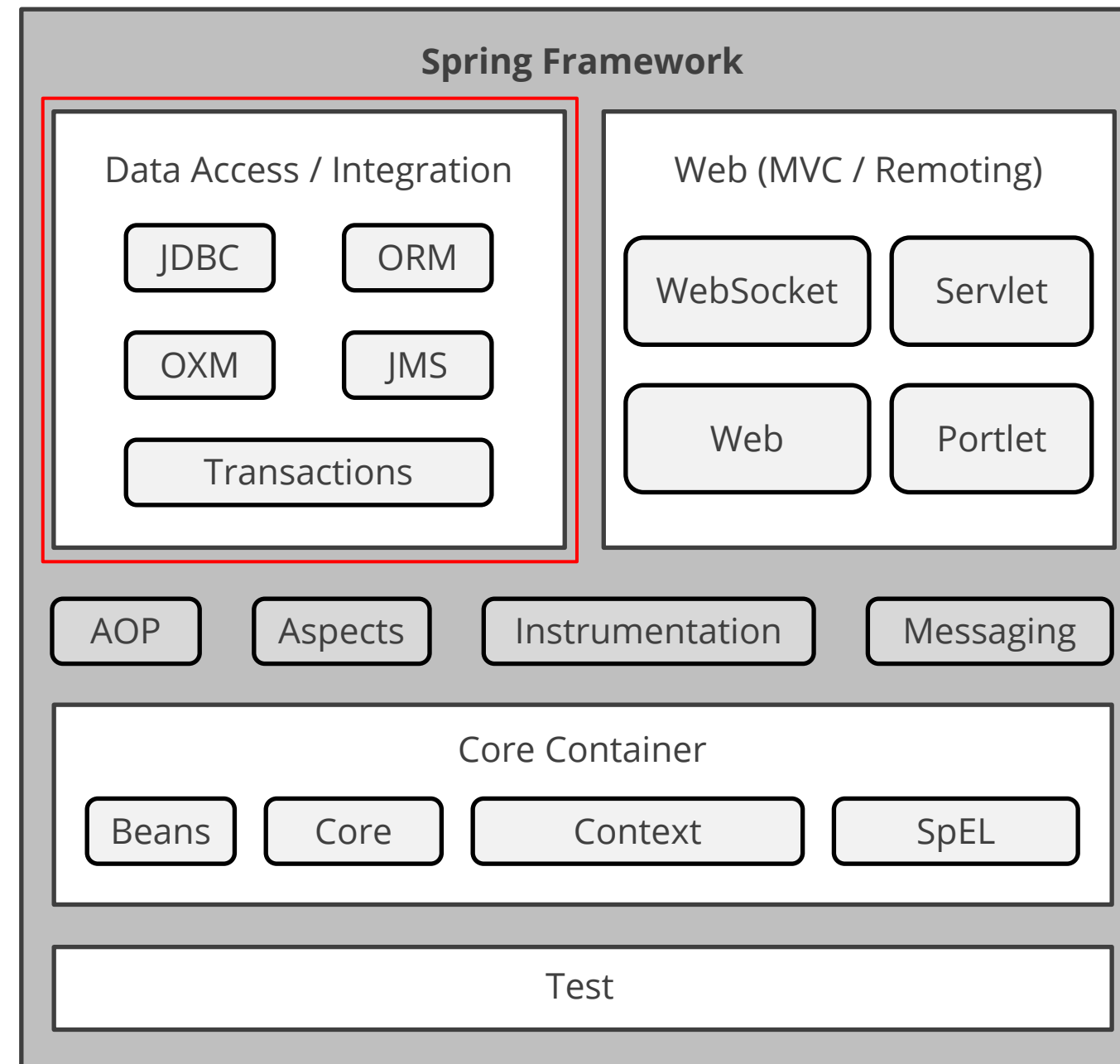


Spring comes with roughly 20 modules.



Spring Architecture

The data access layer consists of the JDBC, ORM, OXM, JMS, and Transactions.



Spring Architecture

The data access or integration layer contains the following:

JDBC

It offers a JDBC abstraction layer that removes the requirement for JDBC-associated coding.

ORM

It provides integration layers for object-relational mapping APIs, including JPA, JDO, iBatis, and Hibernate.

OXM

It offers an abstraction layer supporting Object/XML mapping implementation for Castor, JAXB, JiBX, and Xstream.

Spring Architecture

The data access or integration layer contains the following:

JMS

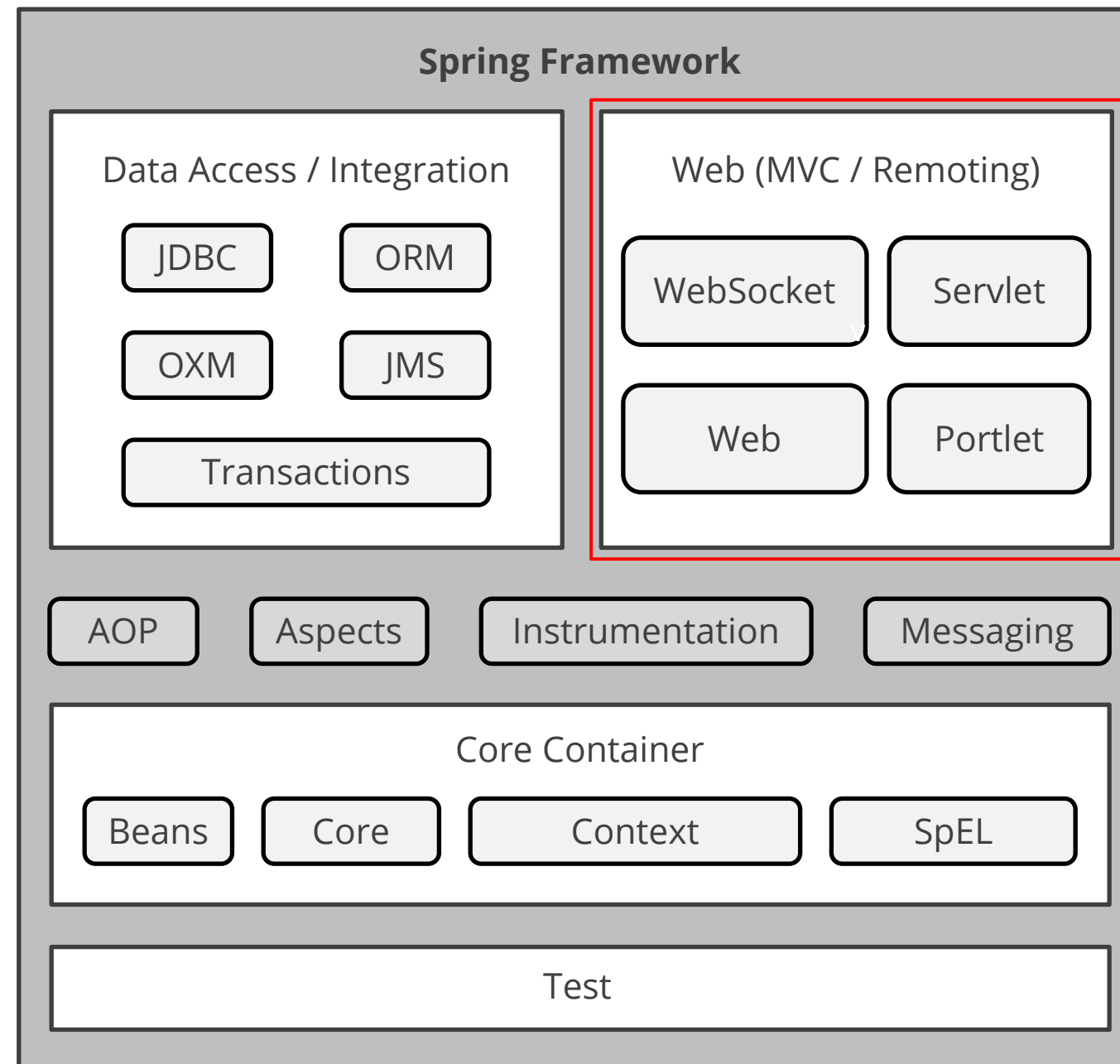
It helps to consume and produce messages.

Transaction Module

It supports declarative and programmatic transaction management for classes implementing special interfaces and all POJOs.

Spring Architecture

The web layer consists of WebSocket, Servlet, Web, and Portlet.



Spring Architecture

The web layer contains the following:

Web

It offers basic web-oriented integration features, like uploading multiple files and initializing IoC containers.

WebSocket

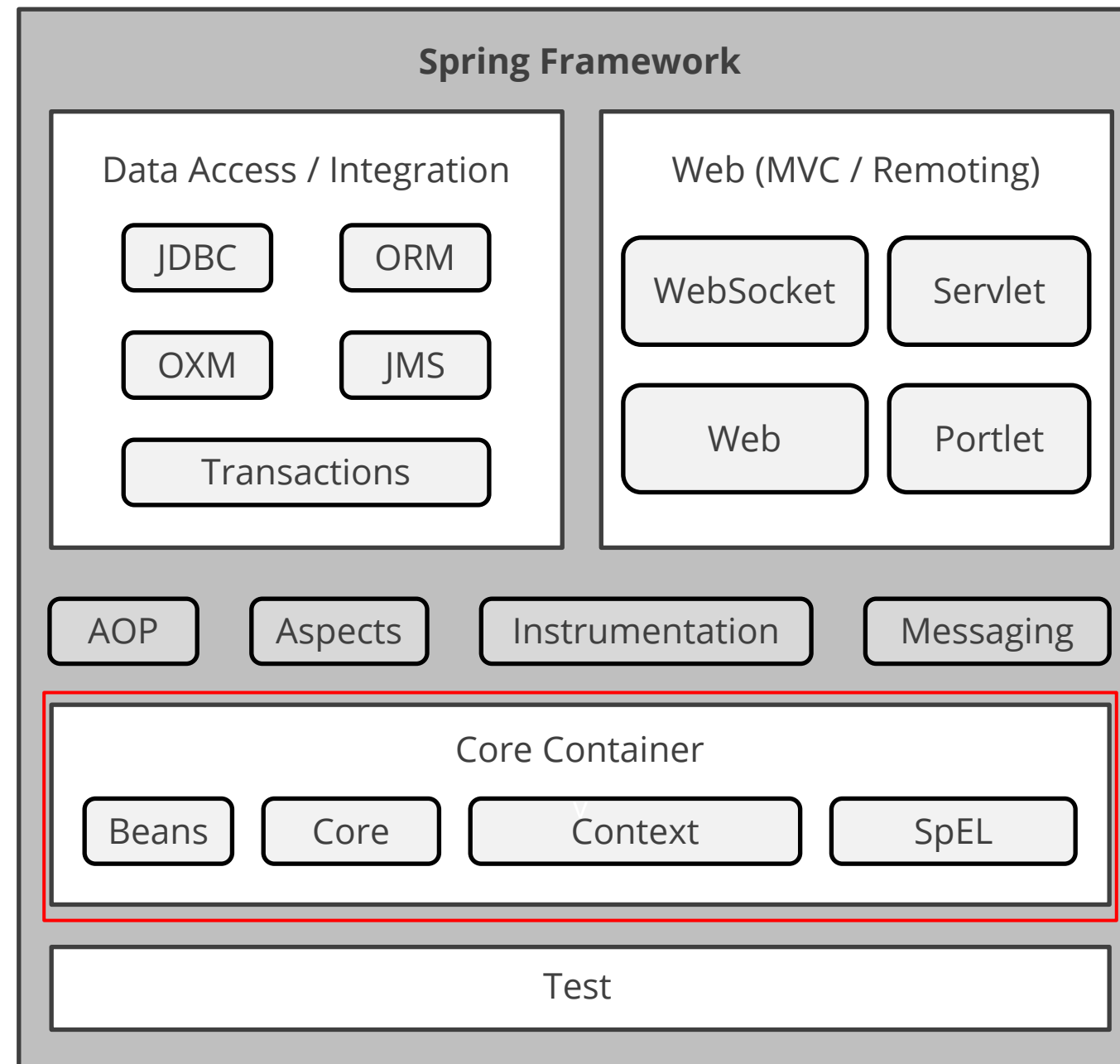
It supports WebSocket-based and is a two-way communication between the client and the server.

Portlet

It offers the MVC implementation for use in a portlet environment and mirrors the working of the Web-Servlet module.

Spring Architecture

Core container consists of Beans, Core, Context, and SpEL.



Spring Architecture

The core container contains the following:

Core

It offers basic parts of the IoC framework and dependency injection features.

Bean

It offers BeanFactory, which is a better implementation of the factory pattern.

Spring Architecture

The core container contains the following:

Context

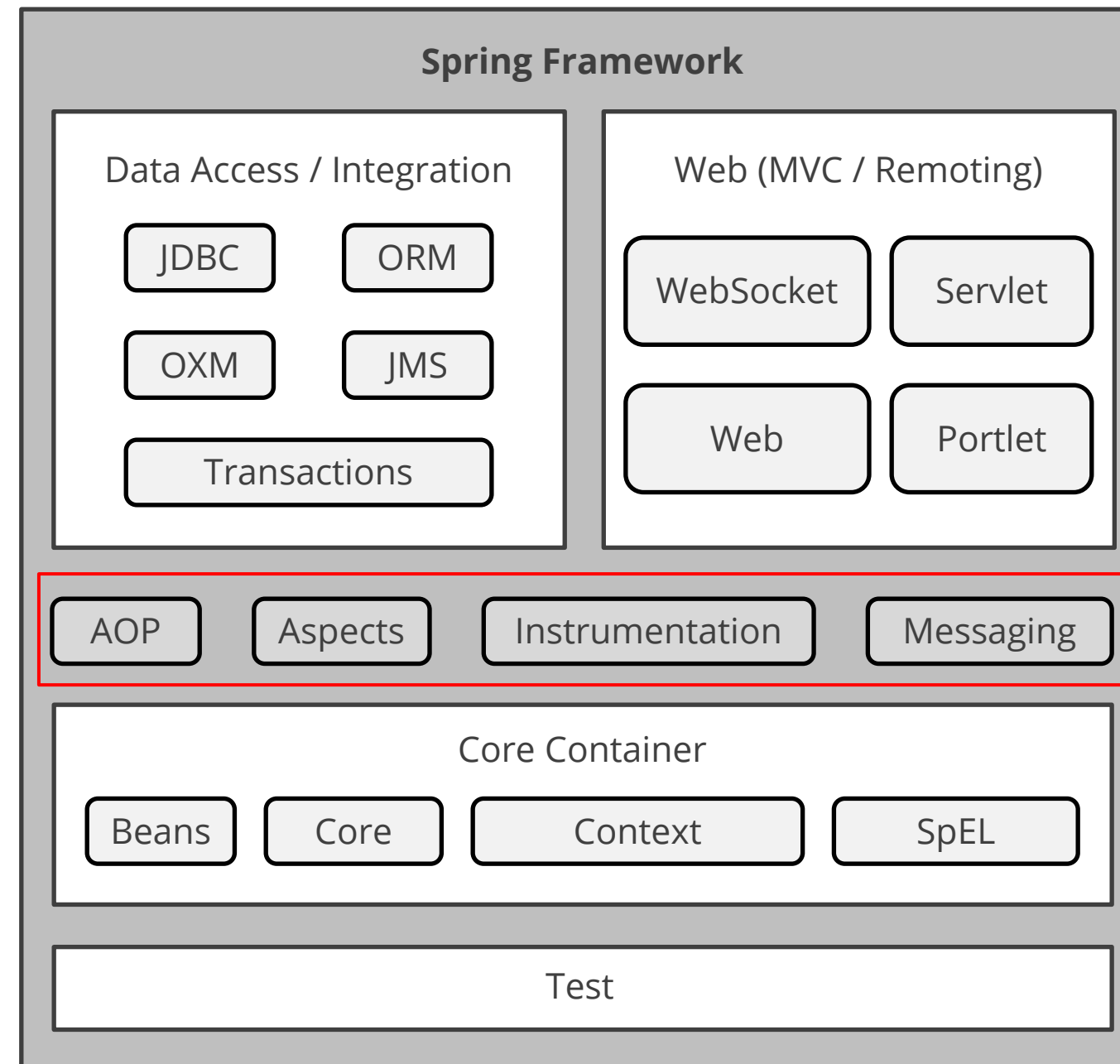
It acts as a base offered by the Core and Beans modules. It also acts as a medium to access any object defined and configured.

SpEL

It offers a detailed expression language to manipulate and query an object graph at runtime.

Spring Architecture

Miscellaneous modules include AOP, Aspects, Instrumentation, and Messaging.



Spring Architecture

The miscellaneous modules contain the following:

AOP

It offers an aspect-oriented programming implementation that helps to define method interceptors and pointcuts.

Aspects

It offers integration with AspectJ, which is a robust and mature AOP framework.

Spring Architecture

The miscellaneous modules contain the following:

Instrumentation

It provides class instrumentation support and class loader implementation used in application servers.

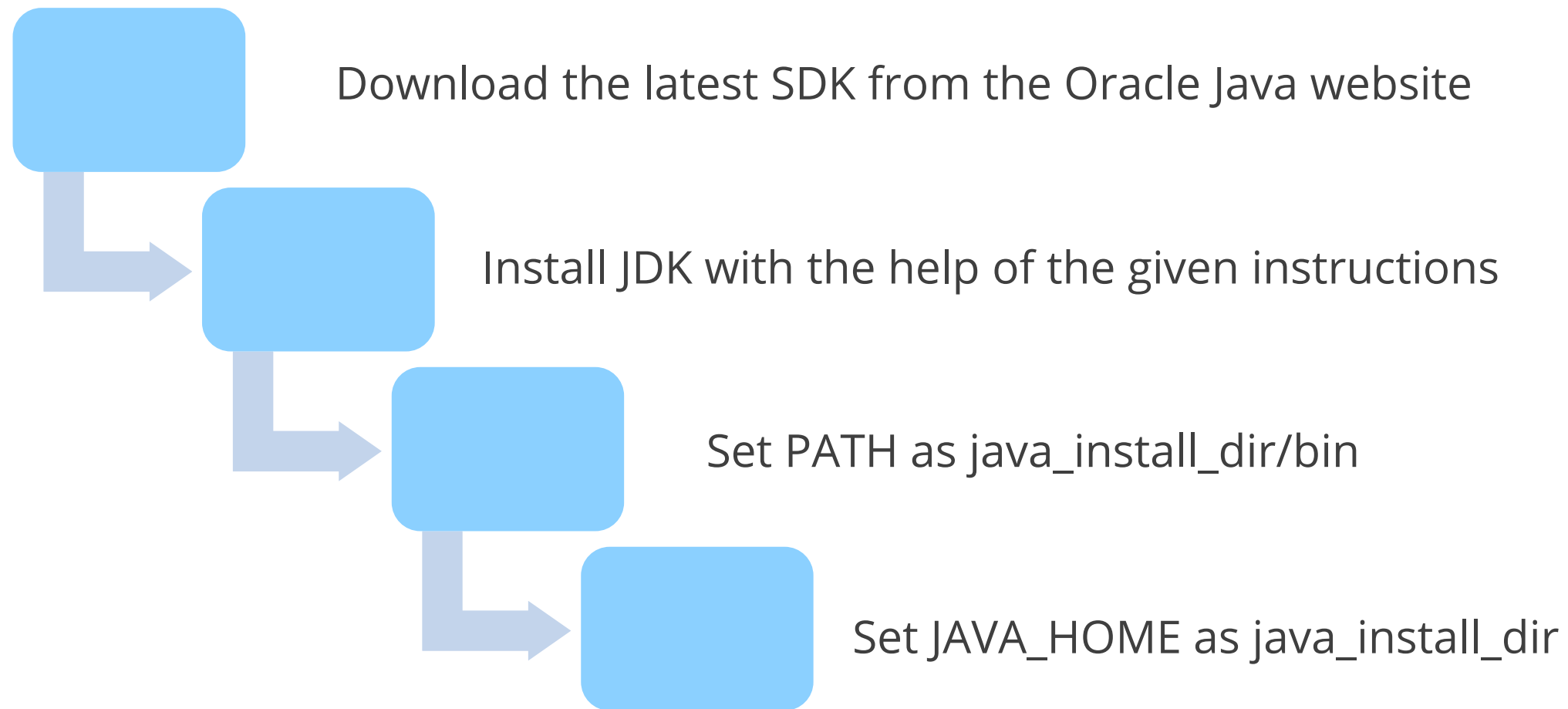
Messaging

It is an annotation programming model for processing and routing STOMP messages from WebSocket clients.

Environment Setup

Setting Up JDK-Java Development Kit

Steps to set up Spring locally:



Setting Up JDK-Java Development Kit

The command used for setting the path:

In Windows

```
PATH=C:\jdk1.x.x_xx\bin;%PATH%  
set JAVA_HOME=C:\jdk1.x.x.xx
```

In Unix

```
setenv PATH  
/usr/local/jdkx.x.x_xx/bin:$PATH  
setenv JAVA_HOME  
/usr/local/jdkx.x.x_xx
```

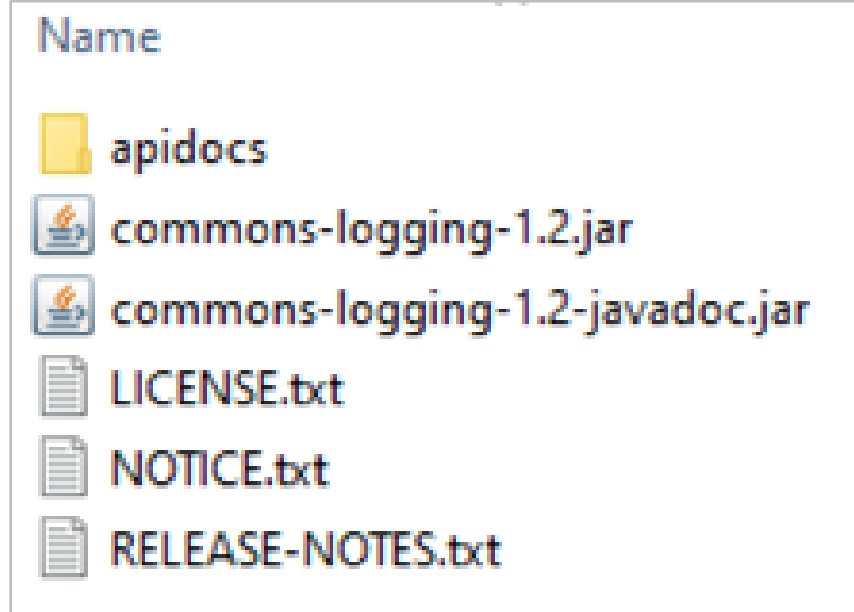
Installing Apache Common Logging API

Below are the steps to install Apache Common Logging API:

Download the latest Apache Common Logging API from:
<https://commons.apache.org/logging/>



Unpack the binary distribution into an accessible location



Ensure that the CLASSPATH variable is set on this directory perfectly

Setting Up Eclipse IDE

To install Eclipse IDE:

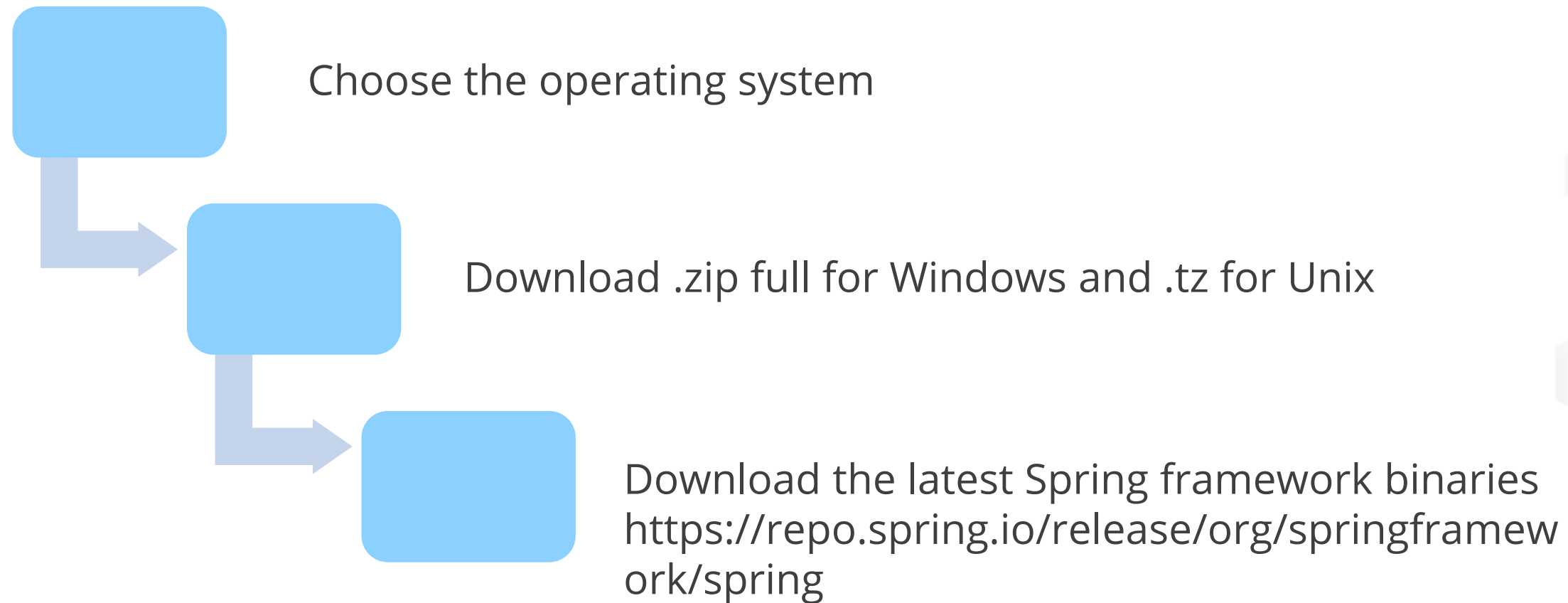
Download the latest version from <https://www.eclipse.org/downloads/>

Unpack the binary distribution into an accessible location

Set the PATH variable accordingly

Setting Up Spring Framework Libraries

Steps for downloading and installing the Spring framework:



Setting Up Spring Framework Libraries

Unzip the downloaded .zip file



Ensure that the CLASSPATH variable on this directory is properly set

Name

- docs
- libs
- schema
- license.txt
- notic.txt
- readme.txt

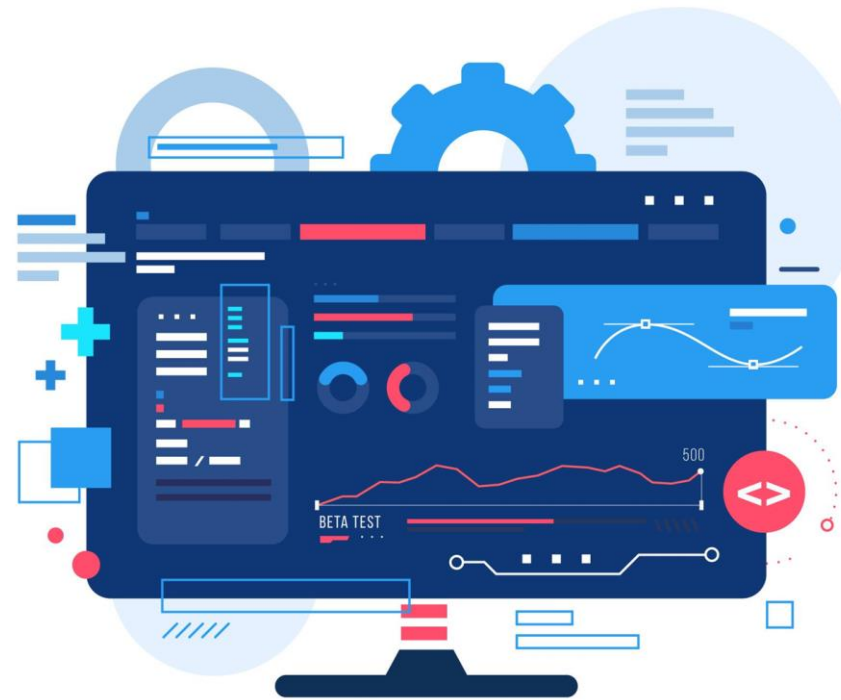
Note

In the case of Eclipse, there is no need to set CLASSPATH because all this is done by Eclipse automatically.

Application Context

Application Context

It is an interface used to deliver configuration information to an application.



Application Context

Spring provides several classes to:

1

Implement `applicationContext`

2

Aid in the use of configuration data in applications

3

Manage the entire lifespan of a `BeanFactory`

Application Context

The class that implements application context must:

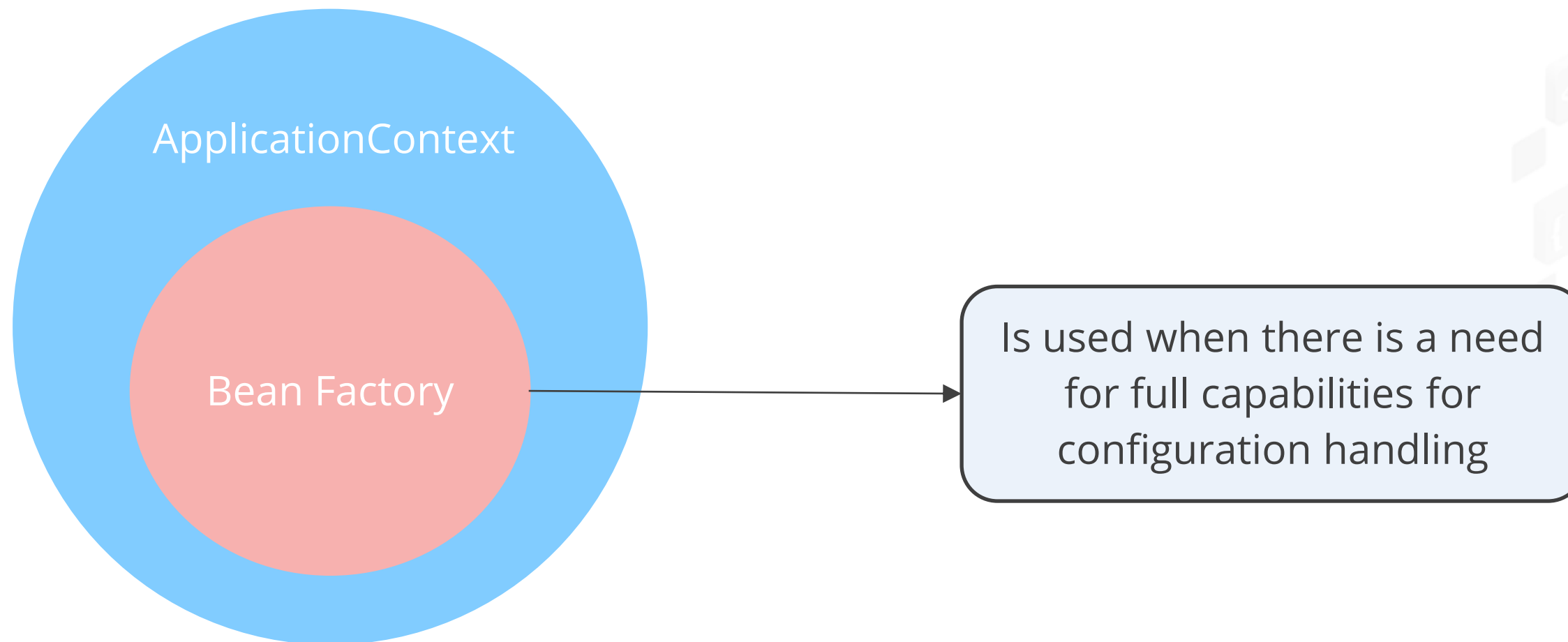
Look for
ApplicationContextAware
beans

Call
setApplicationContext



Bean Factory

Bean factory is a subset of ApplicationContext that offers fewer functionalities.



Bean Factory

Syntax to access an ApplicationContext inside a Java bean:

```
void setApplicationContext(ApplicationContext applicationContext)  
    throws BeansException
```

This method passes concrete objects to ApplicationContext.



To use this syntax, all the configuration information is required.

Bean Factory

Spring provides a utility class called WebApplicationContextUtils.

Syntax for WebApplicationContextUtils:

```
ServletContext servletContext = this.getServletContext();

WebApplicationContext wac =
WebApplicationContextUtils.getRequiredWebApplicationContext(servletCo
ntext);
```


IOC with BeanFactory



Problem Statement:

You have been asked to understand how to implement inversion of control (IOC) using the BeanFactory in the Spring Core framework.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed are:

1. Setting up the development environment
2. Adding dependencies in the **pom.xml** file
3. Creating a bean class
4. Defining the attributes
5. Adding the Getters and Setters
6. Adding bean elements and properties
7. Configuring beans and properties
8. Accessing and running the beans



IOC with Application Context



Problem Statement:

You have been asked to demonstrate the usage of IOC with the application context in a Spring framework project.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed are:


1. Setting up the Maven project
2. Copying files and dependencies
3. Configuring ApplicationContext and retrieving beans
4. Modifying the bean scope
5. Implementing methods for the bean lifecycle



Setter Injection

Setter Injection

Setter-based dependency injection is achieved through two ways:



The container invokes setter methods on beans after calling a no-argument static factory method.



No argument constructor to instantiate bean.

Setter Injection

The setOrderCount() method can be used to set the variable orderCount, as shown:

```
OrderCounter.java

package com.estore;

public class OrderCounter {

    private int orderCount;

    // a setter method to inject the dependency.
    public void setOrderCount(int orderCount) {
        System.out.println("Inside setter method" );
        this.orderCount = orderCount;
    }

    // a getter method to return orderCount
    public int getOrderCount() {
        return orderCount;
    }
}
```

Setter Injection

The configuration for the setter-based injection is as shown:

```
<?xml version = "x.0" encoding = "UTF-8"?>

<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
    http://www.springframework.org/schema/beans/spring-
beans-x.0.xsd">

    <!-- Definition for OrderCount bean -->
    <bean id = "orderCountId" class =
"com.ystore.OrderCount">
        <property name = "orderCount" value = "99"/>
    </bean>

</beans>
```


Setter Injection

The difference in the **Beans.xml** file defined in the constructor DI and setter DI is:

Constructor DI

<constructor-arg> tags

Setter DI

<property> tags

Setter Injection

Below is the example of the setter injection. The main Java file App.java is as shown:

```
public class App{
    public static void main(String[] argos){
        // Using ApplicationContext as IOC
        Container
        ApplicationContext context = new
        ClassPathXmlApplicationContext("beans.xml");
        Order order = context.getBean("order1",
        Order.class);
        System.out.println("Order Count is:
        "+order.getOrderCount());
    }
}
```

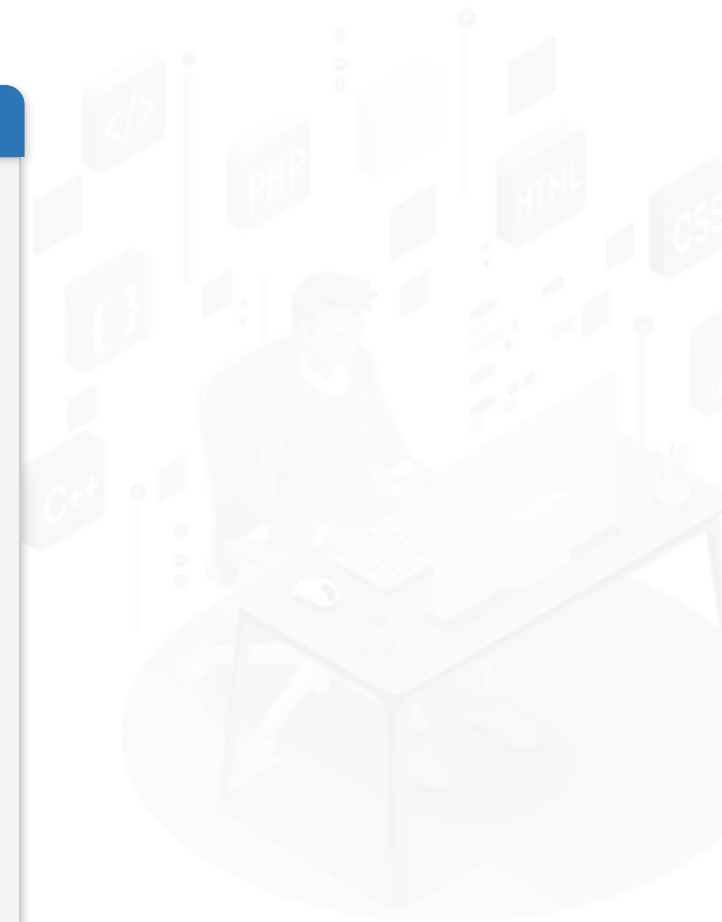
Constructor Injection

Constructor Injection

The classes are provided as parameters to the constructor.

Example:

```
@Component
class Mobile {
    private Brand brand;
    Mobile(Brand brand) {
        Objects.requireNonNull(brand);
        this.brand = brand;
    }
    Brand getBrand() {
        return brand;
    }
    ...
}
```



Constructor Injection

Before version 4.3 of Spring

@Autowired annotation was required to add in the constructor.

Newer versions of Spring

If the class has only one constructor, @Autowired annotation is not mandatory.



In the Mobile class example, there is no need to specify the @Autowired annotation.

Constructor Injection

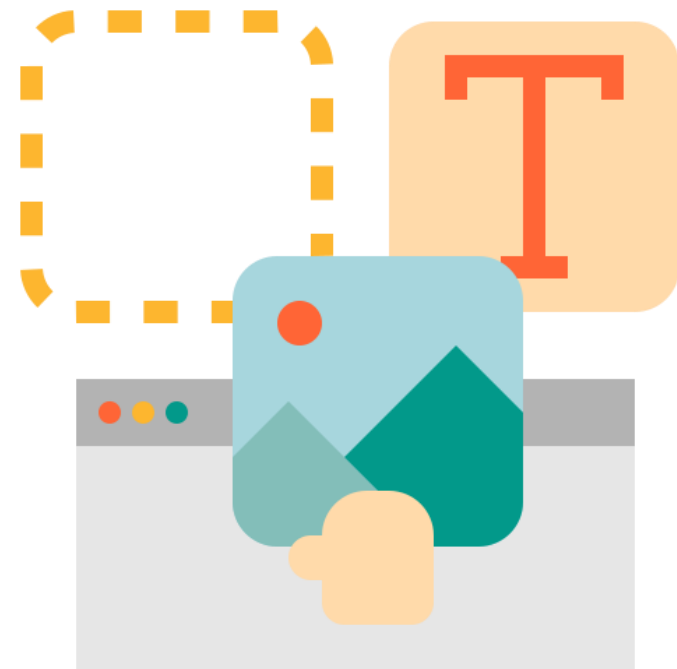
The code below shows the syntax of constructor injection:

```
@Component
class Mobile {
    private Brand brand;
    private Battery battery;
    Mobile(Brand brand) {
        this.brand = brand;
    }
    @Autowired
    Sandwich(Brand brand, Battery battery) {
        this.brand = brand;
        this.battery = battery;
    }
    ...
}
```



Constructor Injection

When there is a class with more than two constructors, explicitly add the `@Autowired` annotation to any of the constructors.



Dependency Injection with Setter and Constructor



Problem Statement:

You have been asked to understand and implement dependency injection using setter and constructor methods in a Spring application.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed are:

1. Creating a Maven project
2. Creating an Address bean
3. Creating the Restaurant bean
4. Configuring the context.xml for beans
5. Writing IOC code in App.java
6. Creating a parameterized constructor
7. Understanding one-to-many relationships



Spring Annotations

Spring Annotations

Annotations are a type of metadata that gives data associated with the program.

Annotations are used to provide supplemental information regarding the program.

Annotations do not affect the working of code or the action of the compiled program directly.



@Required

@Required

This applies to the bean setter method, which indicates that the annotated bean should be populated at the time of configuration with the required property.

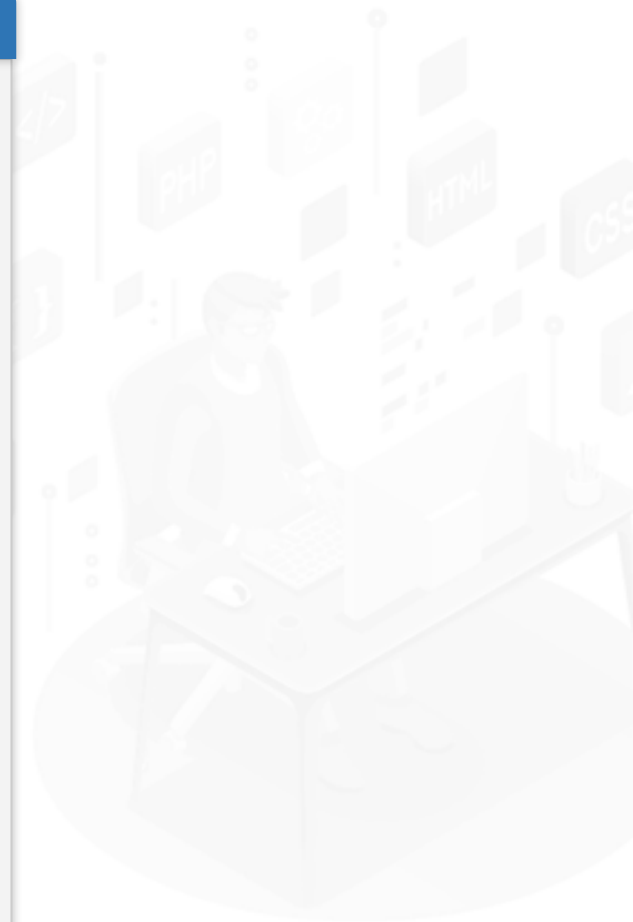
Otherwise, it throws an exception `BeanInitializationException`.



@Required

For example:

```
public class Student
{
    private Integer rollNumber;
    @Required
    public void setRollNumber(Integer rollNumber)
    {
        this.rollNumber = rollNumber;
    }
    public Integer getRollNumber()
    {
        return rollNumber;
    }
}
```

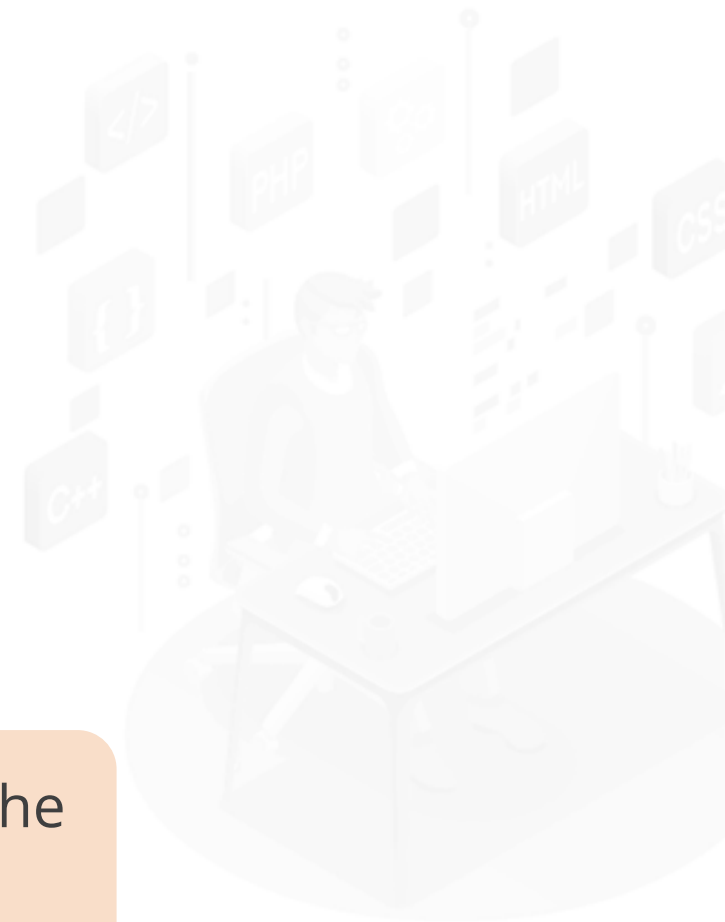


@Autowired

@Autowired

This annotation is employed for auto-wiring Spring beans on instance variables, setter methods, and the constructor.

While using the @Autowired annotation, the Spring container auto-wires the bean by matching the data type.



@Autowired

For example:

```
@Component
public class Student
{
    private Boy boy;
    @Autowired
    public Student(Boy boy)
    {
        this.boy=boy;
    }
}
```



@ComponentScan

@ComponentScan

This annotation is used to scan a package of beans, and it is used with @Configuration annotation.

It is the base package to scan for Spring components that can also be specified.



@ComponentScan

For example:

```
@ComponentScan(basePackages = "com.xx.xx")  
@Configuration  
public class ScanComponent  
{  
    // ...  
}
```

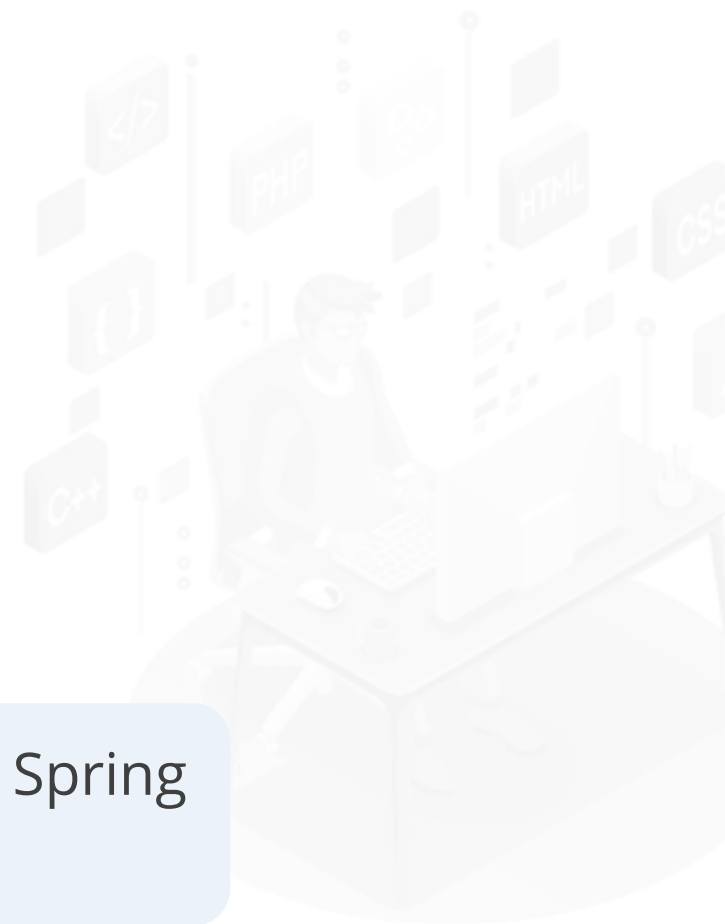


@Bean

@Bean

This is a method-level annotation used as an alternative for XML `<bean>` tag.

It passes the information to the method to produce a bean to be handled by a Spring container.



@Bean

For example:

```
@Bean  
public BeanExample beanExample()  
{  
    return new BeanExample ();  
}
```

Spring offers stereotype annotations that act as markers for any class.

It helps to fulfill a role in an application.



@Component

@Component

This is a class-level annotation used for marking a Java class as a bean.

It is found on the Classpath. Spring picks it up and configures it in the application context as a Spring Bean.



@Component

For example:

```
@Component  
public class Car  
{  
    .....  
}
```



@Controller

@Controller

This is a class-level annotation and an advancement of @Component annotation. It marks a class as a web request handler.

It is also used for serving web pages. By default, it returns a string that indicates which route to redirect. It is employed for the @RequestMapping annotation.

@Controller

For example:

```
@Controller
@RequestMapping("students")
public class StudentsController
{
    @RequestMapping(value = "/{name}", method =
RequestMethod.GET)
    public Teacher getStudentsByName()
    {
        return studentsTemplate;
    }
}
```



@Service

@Service

This is also a class-level annotation. It informs Spring that the class contains business logic.



@Service

For example:

```
package com.xx;  
@Service  
public class TestService  
{  
    public void service1()  
    {  
        //logical code  
    }  
}
```



@Repository

@Repository

This is also a class-level annotation. It is a DAO (Data Access Object) that directly accesses the database.

The repository performs all the operations.



@Repository

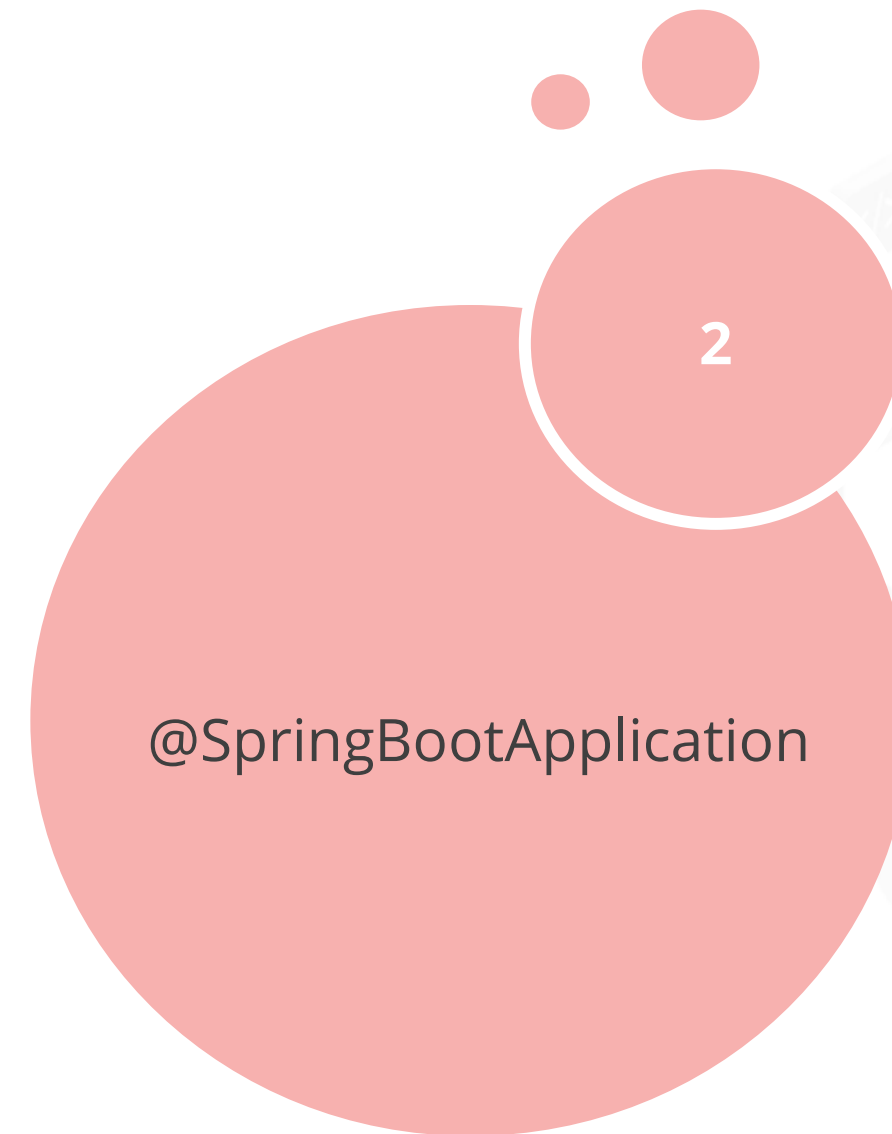
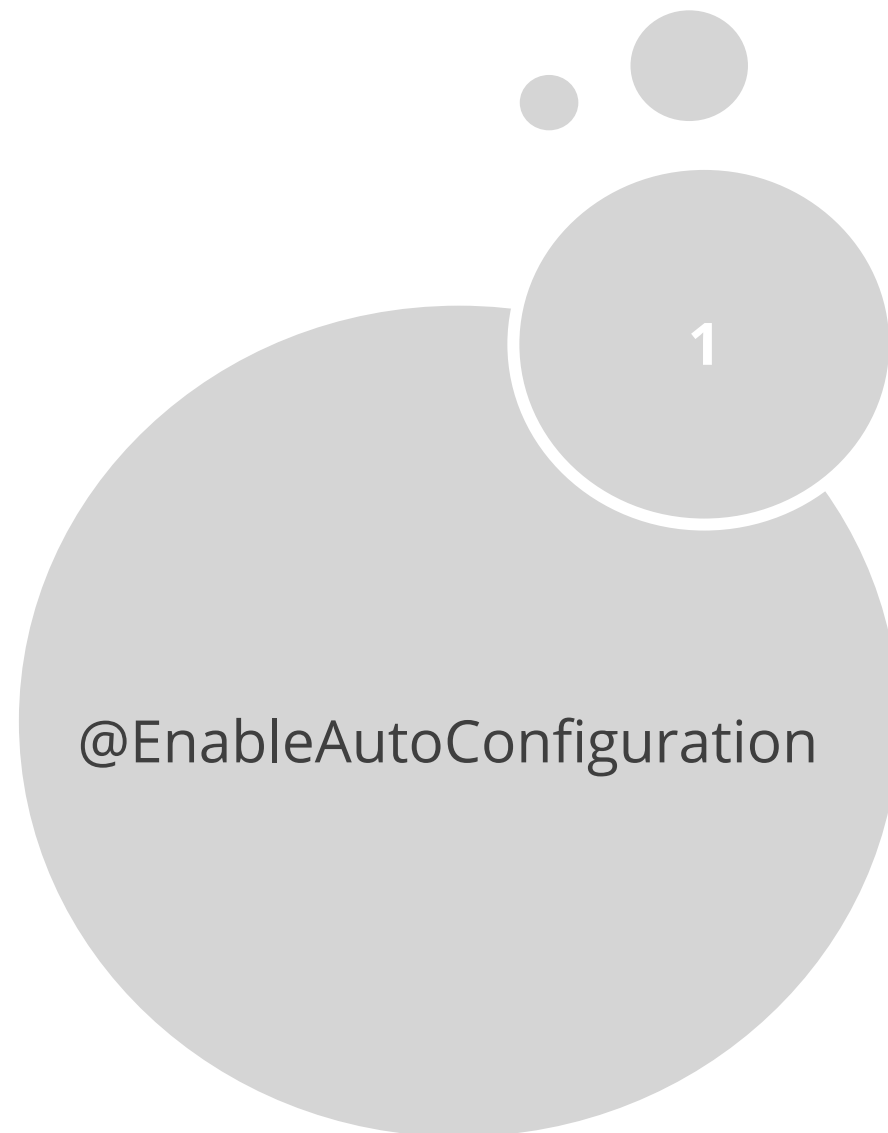
For example:

```
package xx;  
@Repository  
public class TestRepository  
{  
    public void edit()  
    {  
        //persistence code  
    }  
}
```



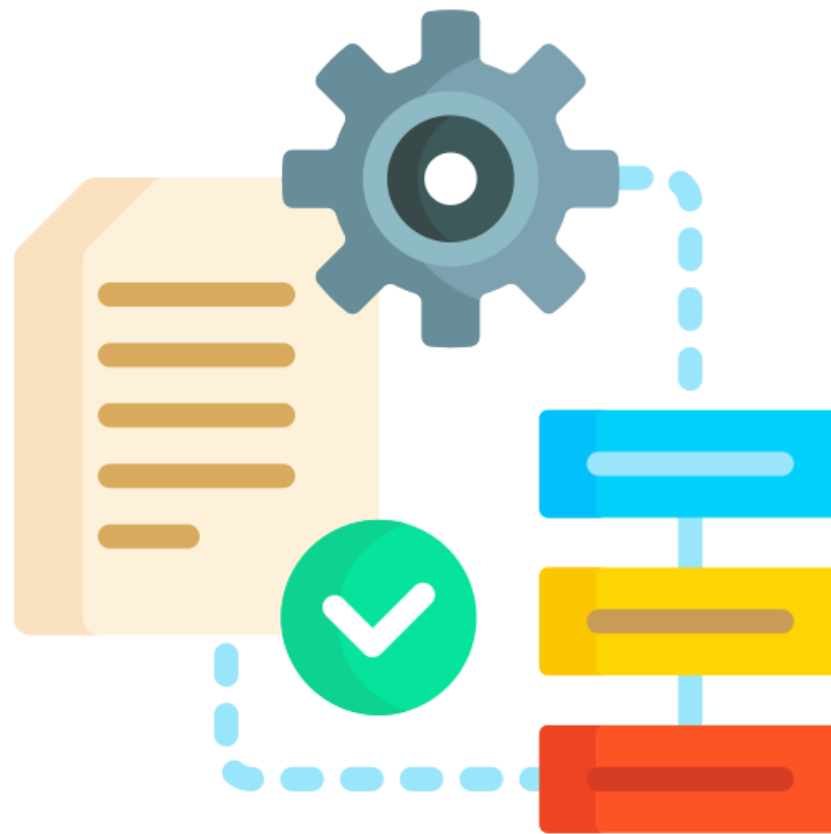
Spring Boot Annotations

There are two major annotations in Spring Boot. They are:



@EnableAutoConfiguration

It helps to auto-configure the bean available in the Classpath and execute the methods.



@SpringBootApplication

It is a combination of three annotations:



The diagram consists of three circles arranged horizontally. The first circle on the left is light blue with a double border and contains the text '@Configuration'. The middle circle is dark blue with a double border and contains the text '@ComponentScan'. The third circle on the right is light red with a double border and contains the text '@EnableAuto Configuration'. In the background, there is a faint illustration of a computer keyboard and some floating code snippets like 'HTML' and 'CSS'.

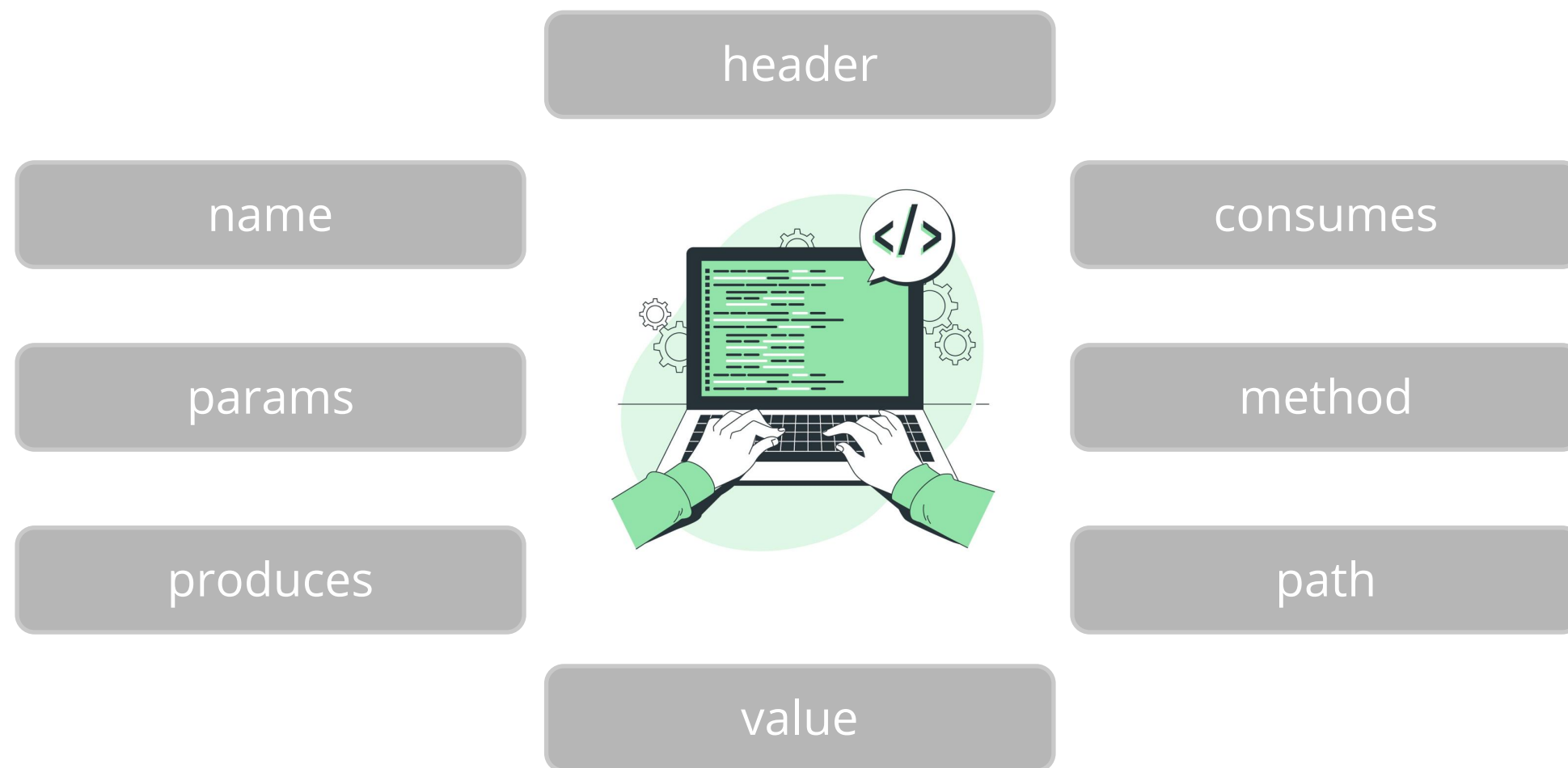
@Configuration

@ComponentScan

@EnableAuto
Configuration

@RequestMapping

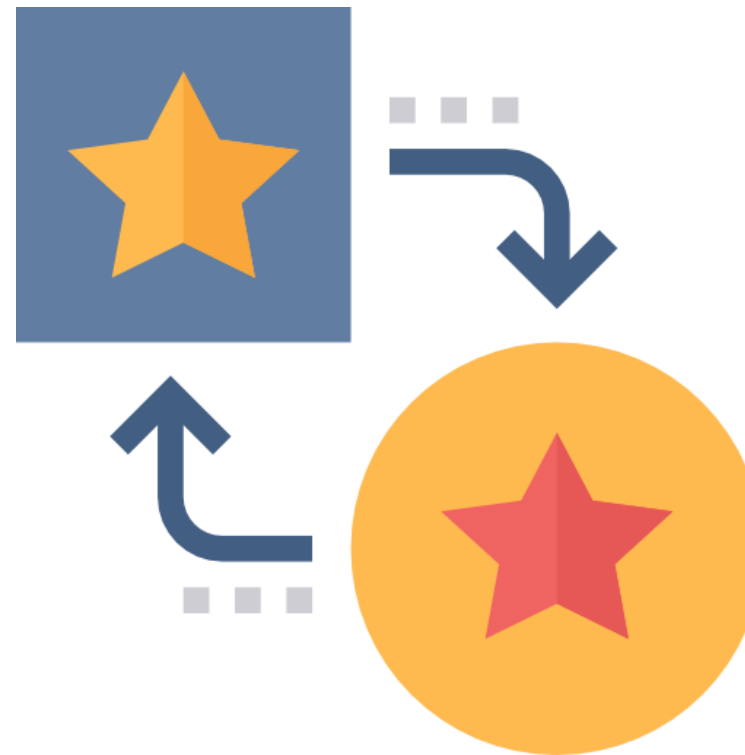
It is employed for mapping the web requests and can be used with the class and method. It contains optional elements like:



The annotations are used in Spring MVC and REST.

@GetMapping Annotation

It helps to map the HTTP GET request and is used to create a web service endpoint.



It is used as an alternative to `@RequestMapping(method = RequestMethod.GET)`.



@PostMapping

It is employed to map the HTTP POST requests and is used to create a web service endpoint.



It is used as an alternative for `@RequestMapping(method = RequestMethod.POST)`.



@PutMapping

It maps the HTTP PUT requests and is employed to create a web service endpoint.

It is used as an alternative for
`@RequestMapping(method = RequestMethod.PUT)`.



@DeleteMapping

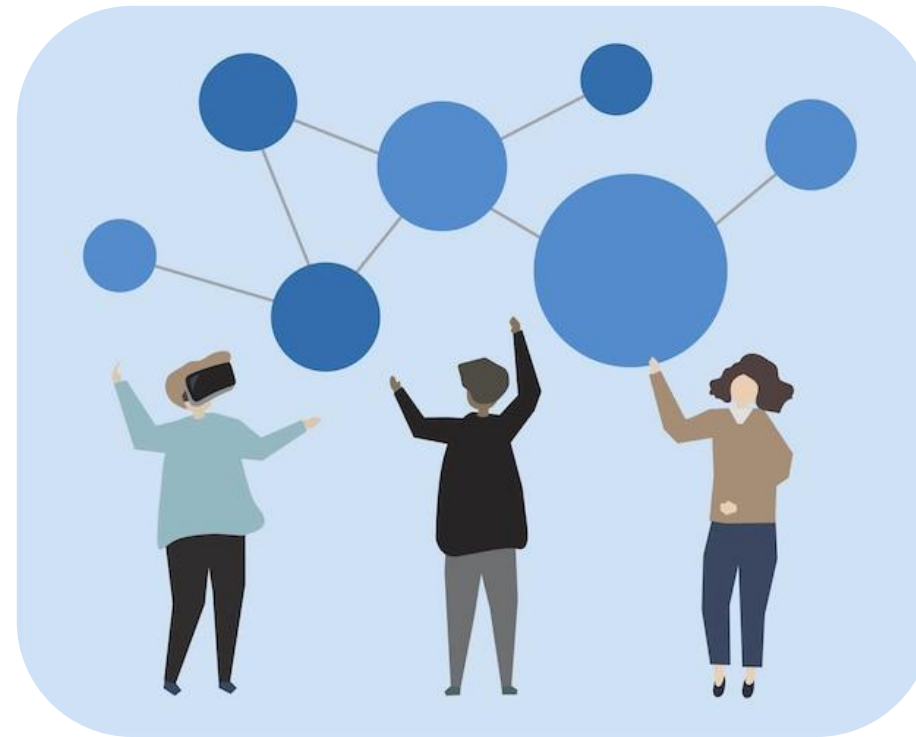
It maps the HTTP DELETE requests and is used to create a web service endpoint that deletes a resource.



It is used as an alternative for `@RequestMapping(method = RequestMethod.DELETE)`.

@PatchMapping

It maps the HTTP PATCH requests on the particular handler method.



It is used as an alternative for `@RequestMapping(method = RequestMethod.PATCH)`.

@RequestBody

It is employed to bind the HTTP request with an object in the parameter of a method.



It takes help from the HTTP MessageConverters to convert the body of the request.

@ResponseBody

It helps to bind the return value of the method to the response body.

It informs the Spring framework to serialize a return object into XML and JSON format.



@PathVariable

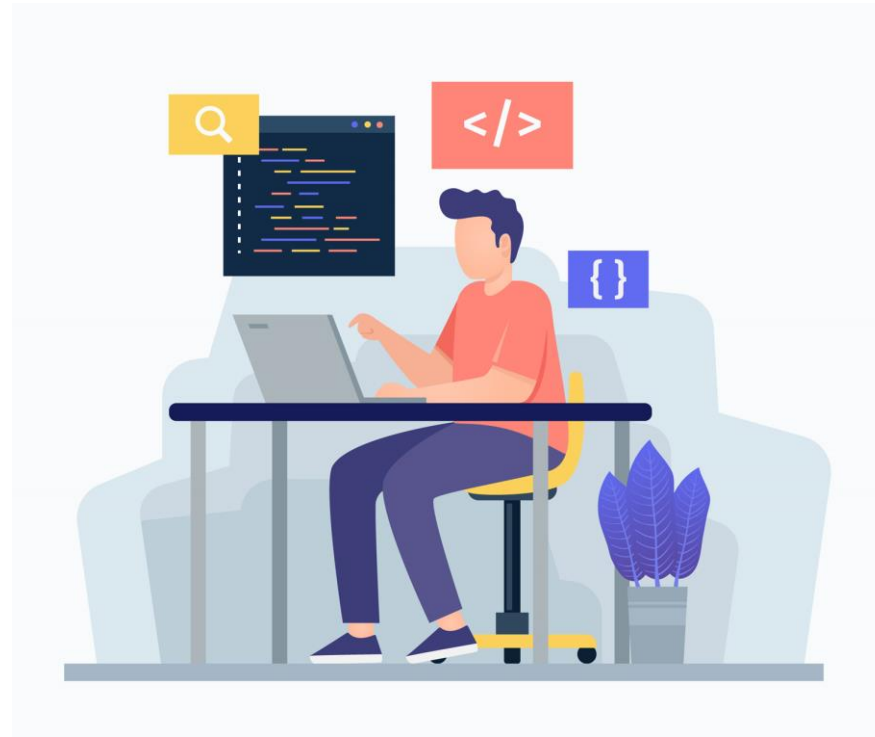
It is used for extracting the values from the URL.



It helps to indicate the default values if the query parameter is not present in the URL.

@RequestHeader

It is used to fetch details regarding the HTTP request headers.



Every detail in the header must specify separate annotations and can be used multiple times in a method.

@RestController

It is a mixture of @Controller and @ResponseBody annotations.

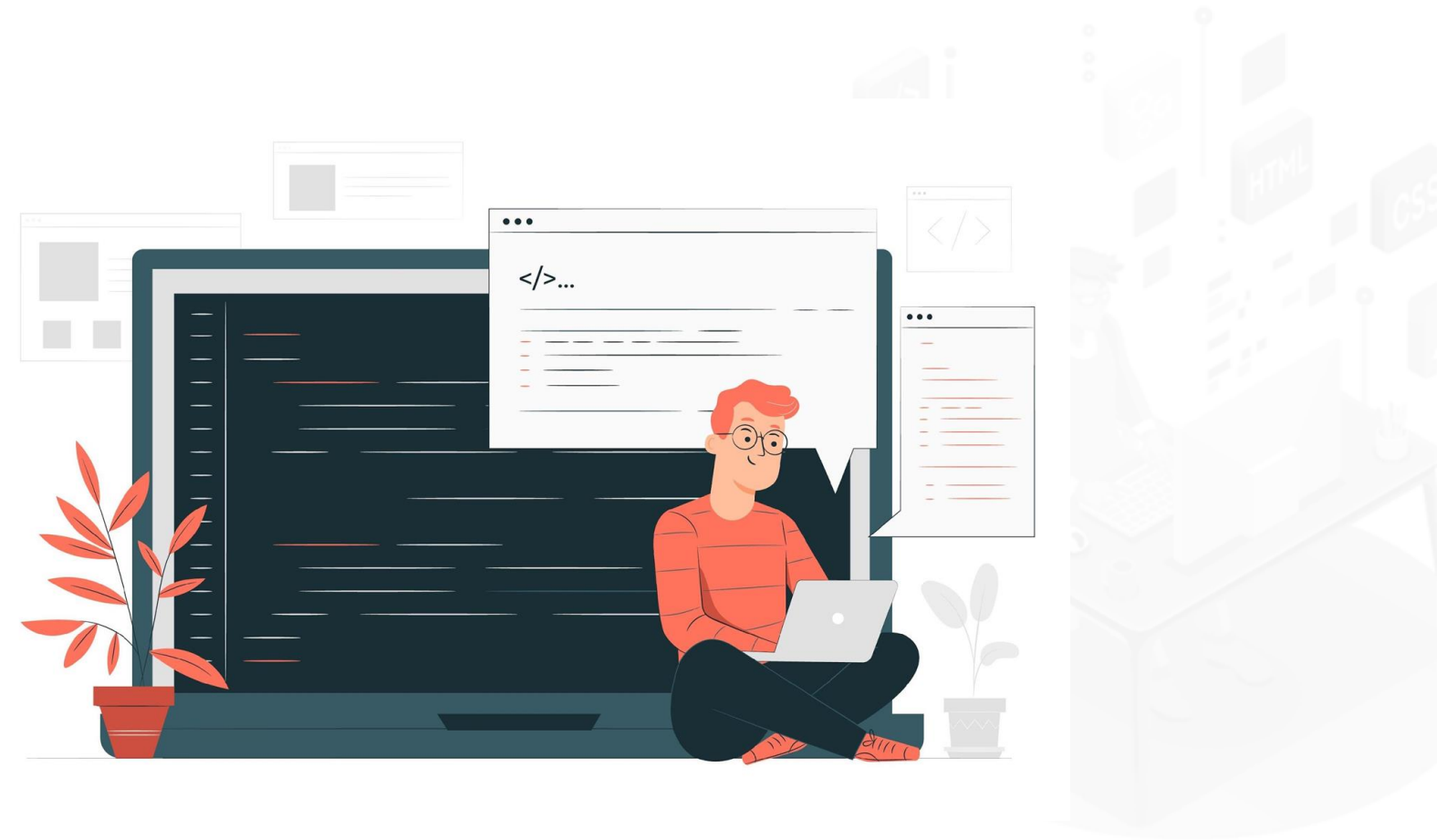


It removes the requirement of annotating every single method with @ResponseBody.

@RequestAttribute

It helps to bind method attributes to the request attribute.

Using @RequestAttribute annotation helps to access objects that are on the server side.



Spring SpEL

Spring SpEL

Spring Expression Language (SpEL):

1

Is a powerful expression of language

2

Provides an option for runtime injection in Spring rather than hard-coding values

3

Injects values into the properties of a constructor or bean arguments that are evaluated at runtime

Spring SpEL

SpEL can be used with annotation-based or XML-based configurations.

SpEL expressions are written in the form `{Expression String}`.



```
<bean id="..." class="...">  
  <property name="..." value = "{ '...' }" />  
</bean>
```

Spring SpEL

An example of bean reference and method invocation with Spring Expression Language:

In the Teacher class, the Student class is referenced.

```
public class Teacher{  
    public String name;  
    public String teacherId;  
    public String address;  
    public String department;  
  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
}
```



Teacher Class

An example of bean reference and method invocation with Spring Expression Language:

```
public String getTeacherId() {
    return teacherId;
}
public void setTeacherId(String teacherId) {
    this.teacherId = teacherId;
}
public String getAddress() {
    return address;
}
public void setAddress(String address) {
    this.address = address;
}
public String getDepartment() {
    return department;
}
public void setDepartment(String department) {
    this.department = department;
}
```

Teacher Class

An example of bean reference and method invocation with Spring Expression Language:

```
public String getStudent(String studentName) {  
    return "Student of [name=" + name +  
    ", teacherId=" + teacherId + ", address=" +  
    address + ", department="+ department + "];"  
}
```



Student Class

An example of bean reference and method invocation with Spring Expression Language:

```
public class Student{
    public String studentId;
    public String studentName;
    public int rollNumber;
    public String teacherName;

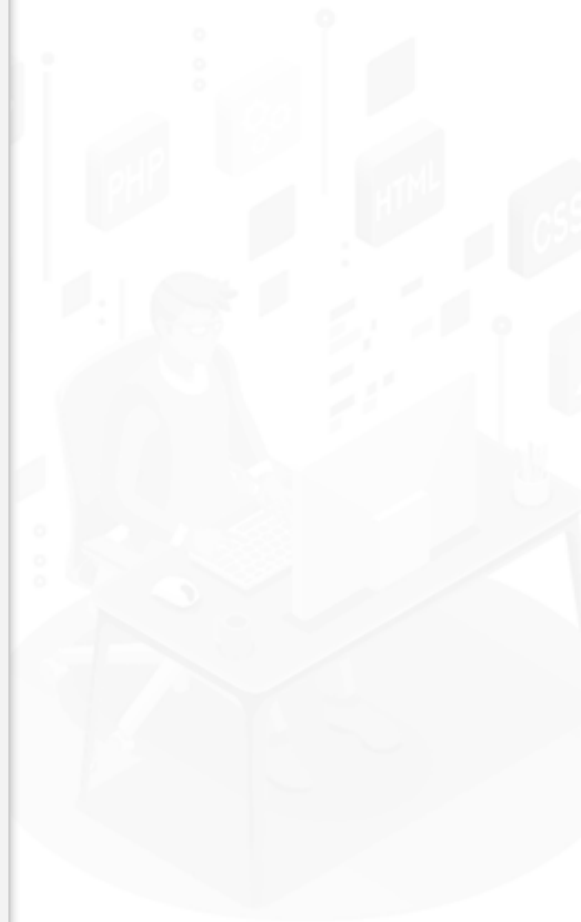
    public String getStudentId() {
        return studentId;
    }
    public void setStudentId(String studentId) {
        this.studentId = studentId;
    }
}
```



Student Class

An example of bean reference and method invocation with Spring Expression Language:

```
public String getStudentName() {  
    return studentName;  
}  
public void setStudentName(String studentName) {  
    this.studentName = studentName;  
}  
public int getRollNumber() {  
    return rollNumber;  
}  
public void setRollNumber(int rollNumber) {  
    this.rollNumber = rollNumber;  
}  
public String getTeacherName() {  
    return teacherName;  
}  
public void setTeacherName(String teacherName) {  
    this.teacherName = teacherName;  
}  
}
```



Spring SpEL

The XML Configuration for the code is:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">
```

Spring SpEL

The XML file in the teacherName Bean properties is provided with values using literal expressions.

```
<bean id="teacher" class="com.xx.Teacher">
  <property name="name" value = "John Watson" />
  <property name="teacherId" value = "101" />
  <property name="address" value = "2144 Redwood Shores" />
  <property name="department" value = "ComputerScience"/>
</bean>
```

Spring SpEL

SpEL gives the studentId reference to employee bean teacher property.

```
<bean id="student" class="com.xx..Student">
    <property name="studentId" value = "201" />
    <property name="studentName" value = "Fionna" />
    <!-- Bean reference through SpEL -->
    <property name="teacherName" value =
    "#{teacher.name}" />
</bean>
</beans>
```

Inheritance Relationship in Spring Core



Problem Statement:

You have been asked to demonstrate inheritance in Spring Core through the creation of bean classes and utilizing Inversion of Control (IOC).

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed are:

1. Creating a Maven project
2. Copying files and dependencies
3. Creating the FoodItem bean
4. Creating the Pizza bean
5. Configuring the context.xml for beans
6. Writing IOC code in App.java



Key Takeaways

- Spring is a powerful Java corporate application development framework.
- The Spring framework can be used to create any type of Java program.
- Dependency Injection is the basic functionality given by Spring IOC.
- The application context is an interface used to deliver configuration information to an application.
- Annotations are a type of metadata that gives data associated with the program.



TECHNOLOGY

Thank You