Building Web Applications Using the Spring Framework

Aesthetic: Tarkeshwar Barua

Monolithic Architecture

- *An application in which we implement all the responsibilities in the same executable artifact.*
- *You can see this as one application that fulfills all the use cases. The responsibilities can sometimes be implemented within different modules to help the application be more comfortable to maintain.*
- *The logic of one can't be separated from the logic of others at run time.*
- *Monolithically architectures offer less flexibility for scaling.*

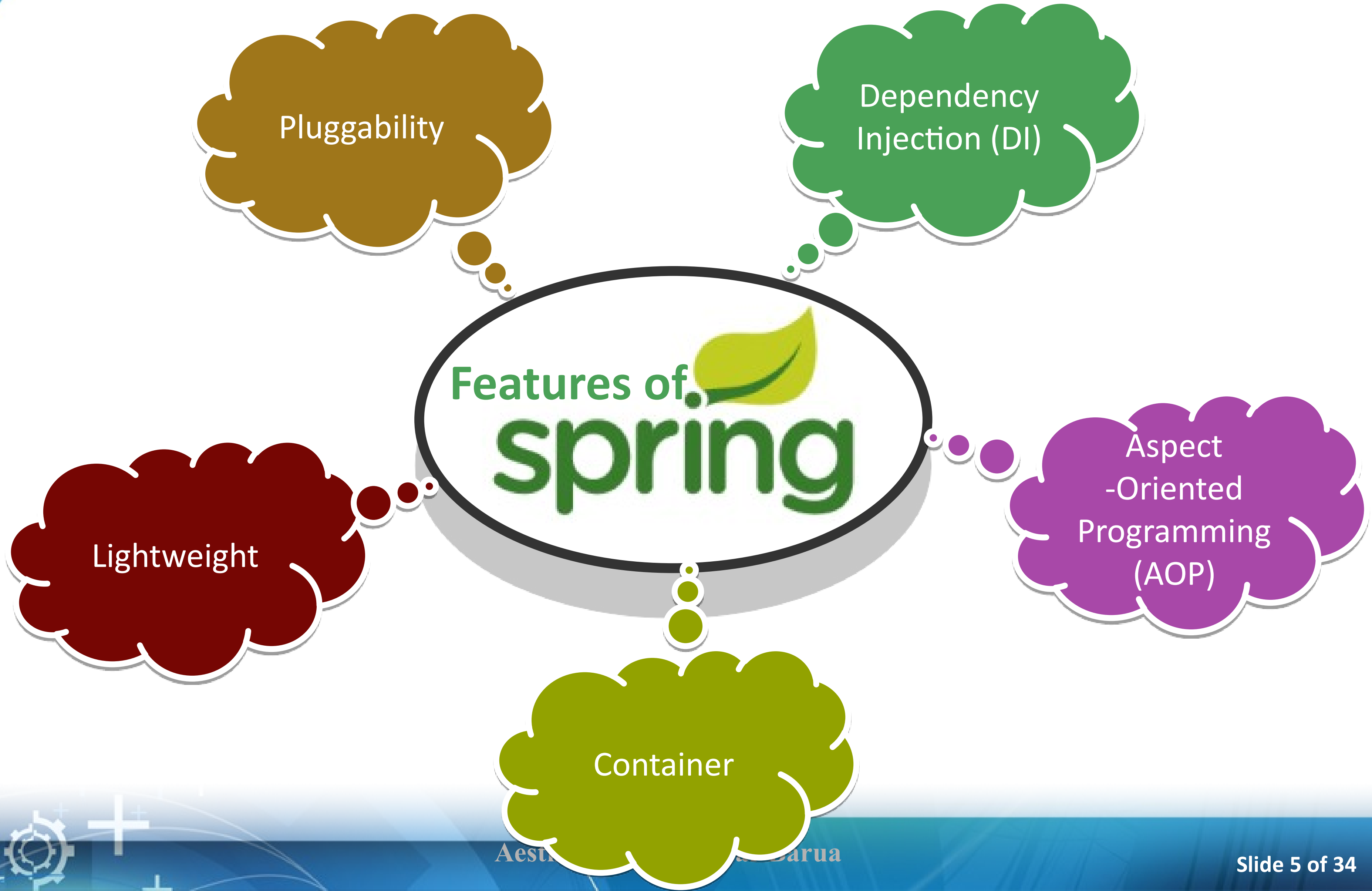
Micro Service Architecture

- *A microservice system has the responsibilities implemented within different executable artifacts.*
- *You can see the system as being formed of multiple applications that execute at the same time and communicate between them when needed via the net-work.*
- *While this offers more flexibility for scaling, it introduces other difficulties.*
- *We can enumerate here latencies, security concerns, network reliability, distributed persistence, and deployment management.*

Introducing the Spring Framework

- *Aims to overcome the application problems by enabling the use of simple JavaBeans (POJOs) to implement the business logic.*
- *Enables the developers to create and test applications easily.*
- *Aims to minimize the dependency of application code on its framework.*
- *Enables the use of simplicity and ease of testability in standalone applications.*

Features of Spring



Comparing Spring with Struts and EJB

spring

EJB

Is a popular non-standard open-source framework developed by Interface21 Inc employee Rod Johnson in 2003.

- ❑ Enables you to implement only those features that you require for an application.

- ❑ Is a specification defined by the Java Community Process (JCP) and supported by all the major J2EE vendors.

- ❑ Provides a set of prepackaged features, most of which you may not need.

Comparing Spring with Struts and EJB (Contd.)

Benefits of Spring MVC over the Struts framework

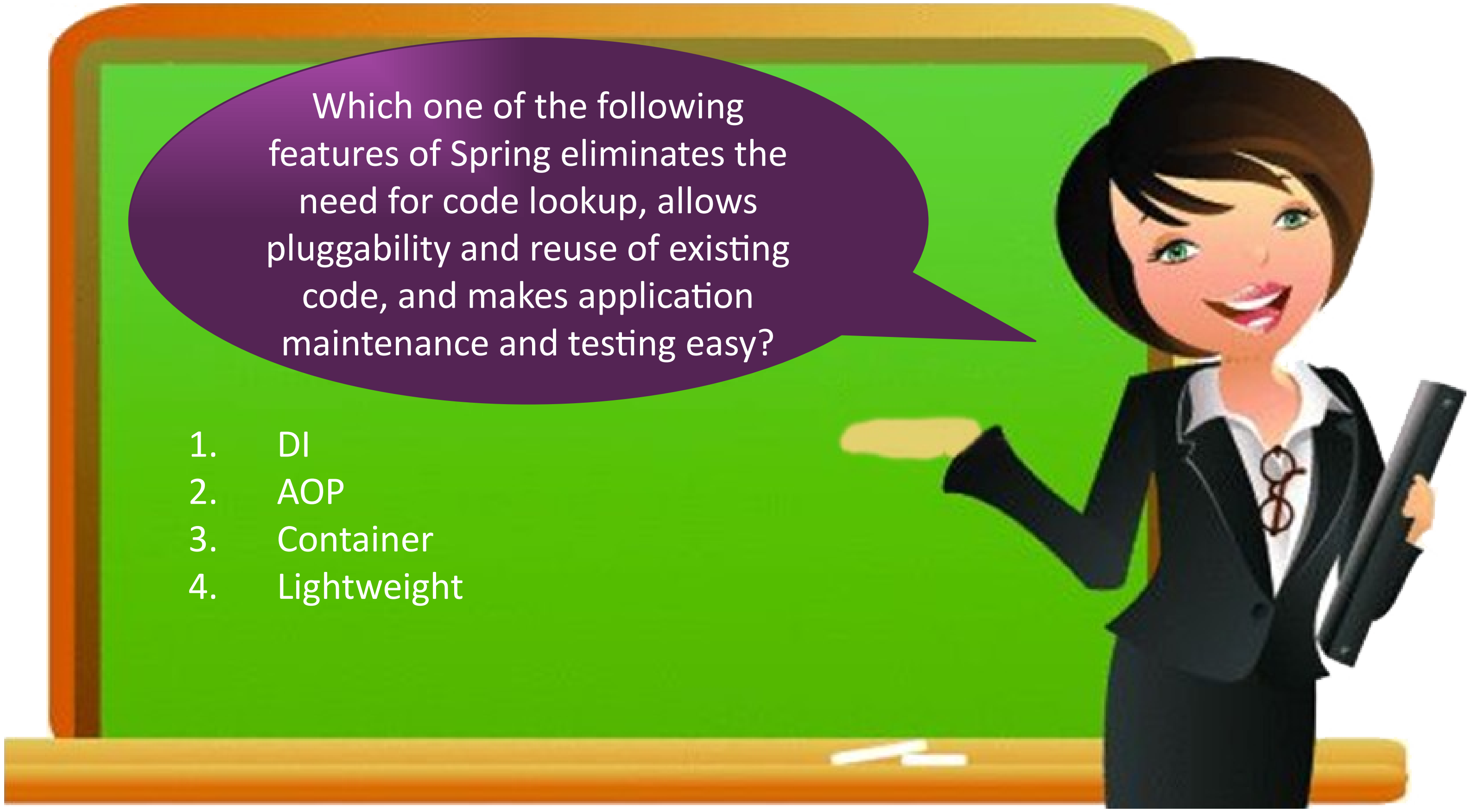
- *Provides a clear separation between the controllers, models, and views.*
- *Is highly flexible, unlike Struts that does not implement interfaces and forces your Action and Form objects into concrete inheritance.*
- *Provides controllers for handling user requests.*
- *supports several view technologies, such as JSP, Velocity, Tiles, and JSF, Thymeleaf, Free Marker.*
- *Provides controllers that can be easily configured through DI, just like any other application object.*

Comparing Spring with Struts and EJB (Contd.)

Benefits of Spring MVC over the Struts framework

- ***Allows the development of application that are easier to test than the Struts MVC Web applications.***
- ***Allows the implementation of validators as application objects, which are not dependent on the Spring API.***
- ***Does not impose dependencies on controllers.***
- ***Provides convenience controller implementations that you can extend, if required, by the application.***
- ***Allows you to integrate easily with the existing technologies.***

Just a Minute

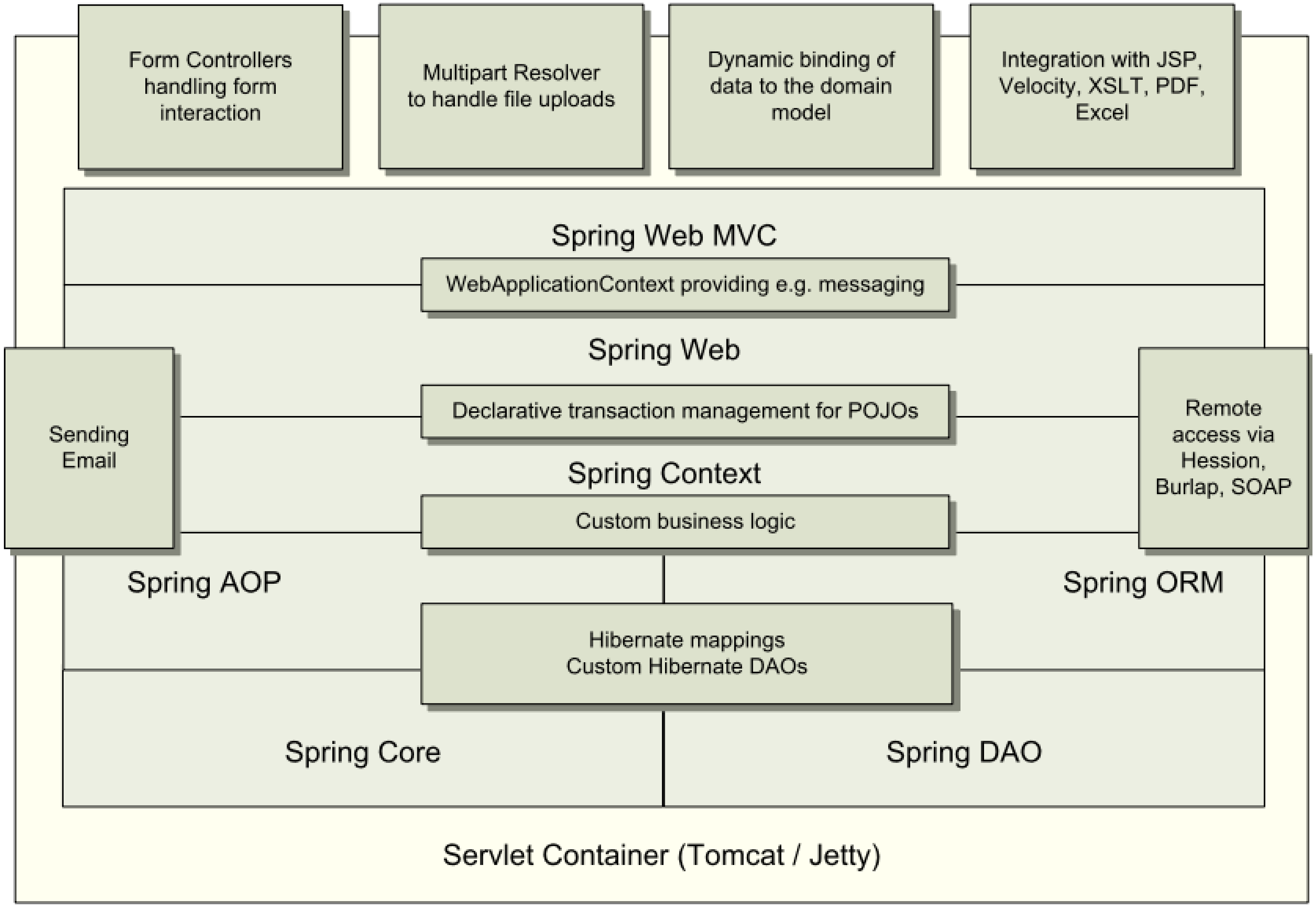


Which one of the following features of Spring eliminates the need for code lookup, allows pluggability and reuse of existing code, and makes application maintenance and testing easy?

1. DI
2. AOP
3. Container
4. Lightweight

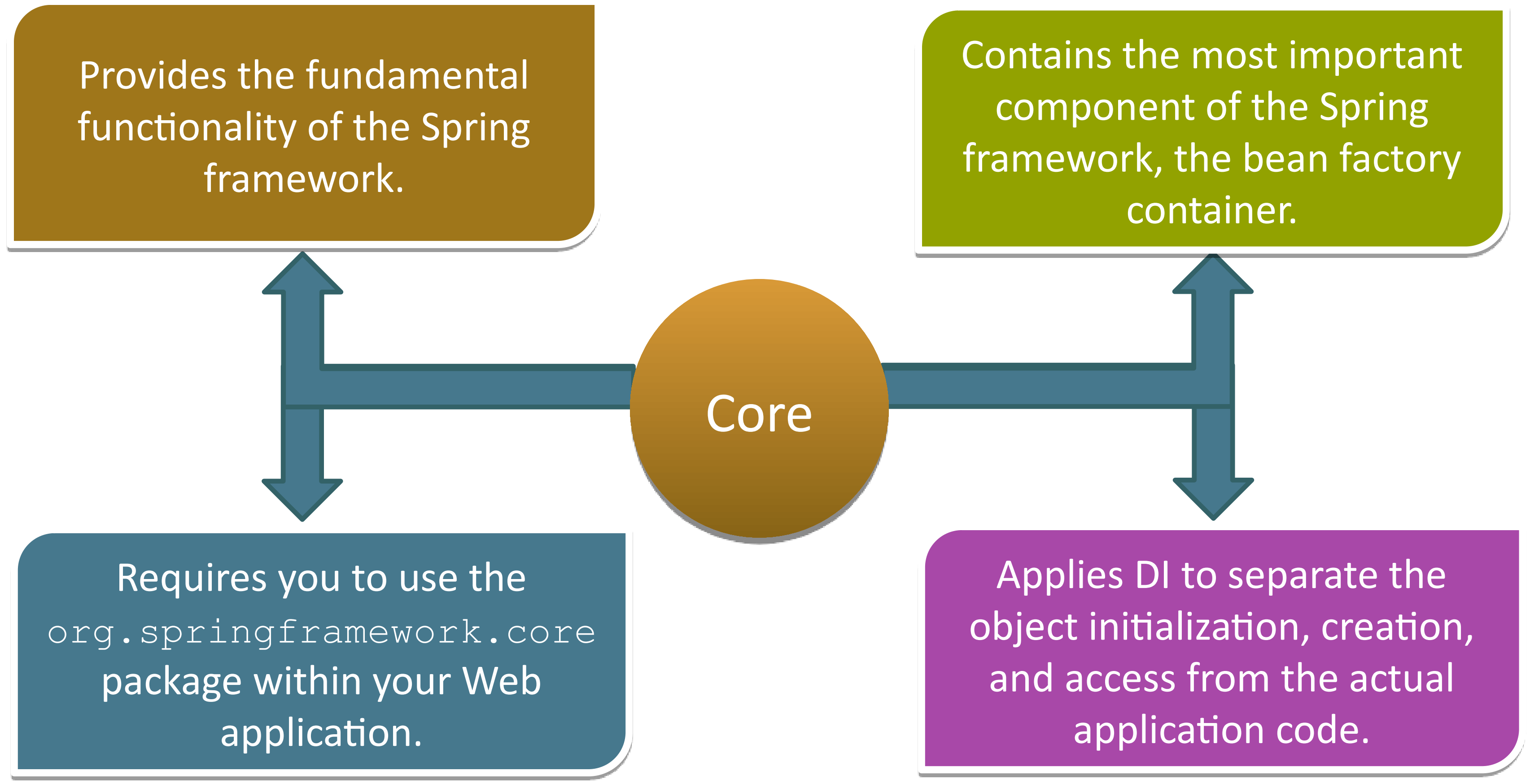
Answer: 1. DI

Introducing the Spring Architecture



Form Controllers handling form interaction
Multipart Resolver to handle file uploads
Dynamic binding of data to the domain model
Integration with JSP, Velocity, XSLT, PDF, Excel
Spring Web MVC
WebApplicationContext providing e.g. messaging
Spring Web
Sending Email
Declarative transaction management for POJOs
Remote access via Hession, Burlap, SOAP
Spring Context
Custom business logic
Spring AOP
Spring ORM
Hibernate mappings
Custom Hibernate DAOs
Spring Core
Spring DAO
Servlet Container (Tomcat / Jetty)

Core


Provides the fundamental functionality of the Spring framework.
Contains the most important component of the Spring framework, the bean factory container.
Core
Requires you to use the org.springframework.core package within your Web application.
Applies DI to separate the object initialization, creation, and access from the actual application code.

Spring Core

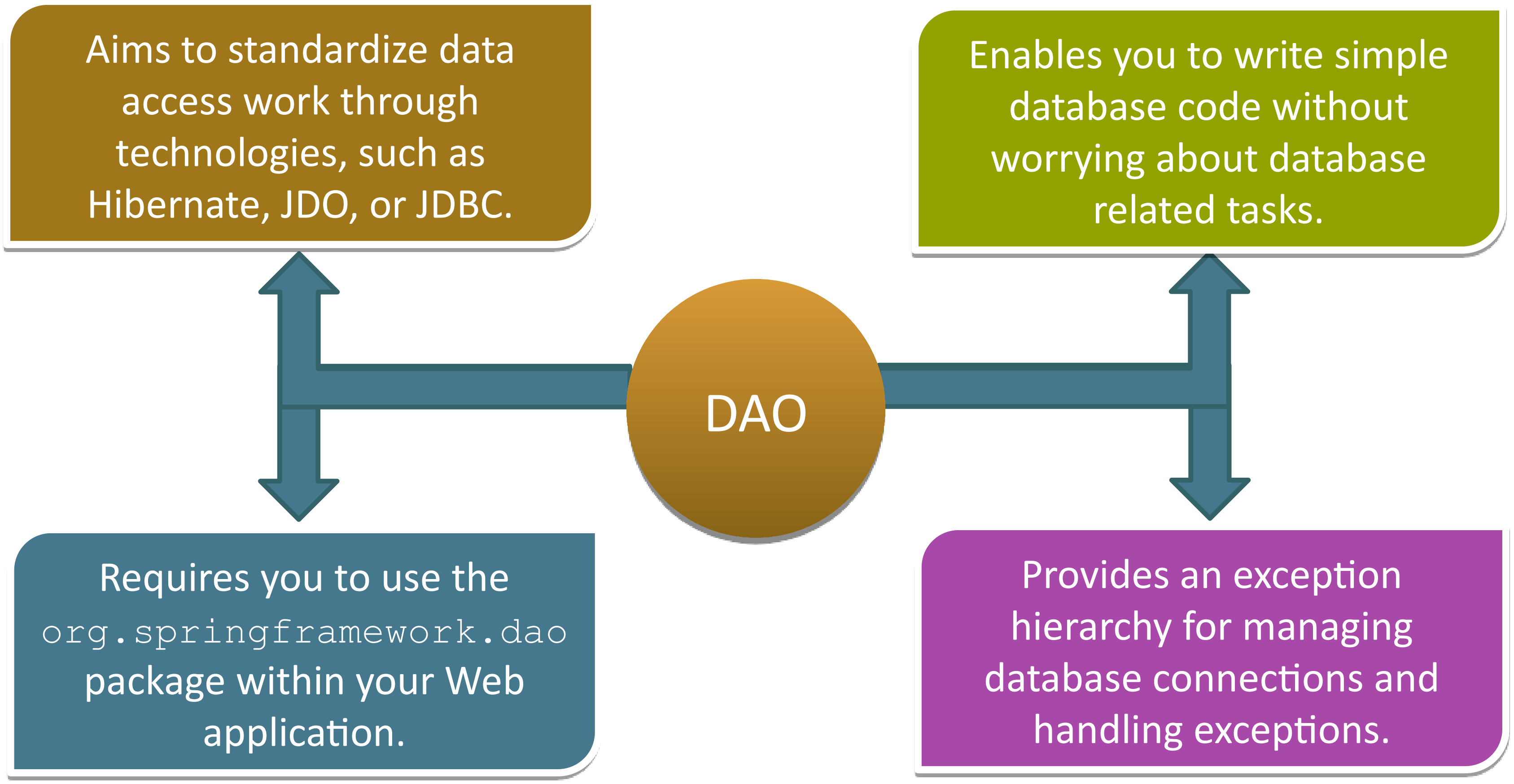
- *Spring Framework does not force you to use everything within it; it is not an all-or-nothing solution.*
- *Existing front-ends built using Thymeleaf, React, Angular, WebWork, Struts, Tapestry, JSF or other UI frameworks can be integrated perfectly well with a Spring-based middle-tier, allowing you to use the transaction features that Spring offers.*
- *The only thing you need to do is wire up your business logic using an **ApplicationContext** and integrate your web layer using a **WebApplicationContext***

Spring Core

Spring Core: *The Core package is the most fundamental part of the framework and provides the IoC and Dependency Injection features.*
The basic concept here is the BeanFactory, which provides a sophisticated implementation of the factory pattern which removes the need for programmatic singletons and allows you to decouple the configuration and specification of dependencies from your actual program logic.

Spring context: *The Context package build on the solid base provided by the Core package: it provides a way to access objects in a framework-style manner in a fashion somewhat reminiscent of a JNDI-registry.*
The context package inherits its features from the beans package and adds support for internationalization (I18N) (using for example resource bundles), event-propagation, resource-loading, and the transparent creation of contexts by, for example, a servlet container.

DAO



Aims to standardize data access work through technologies, such as Hibernate, JDO, or JDBC.

Enables you to write simple database code without worrying about database related tasks.

Requires you to use the `org.springframework.dao` package within your Web application.

Provides an exception hierarchy for managing database connections and handling exceptions.

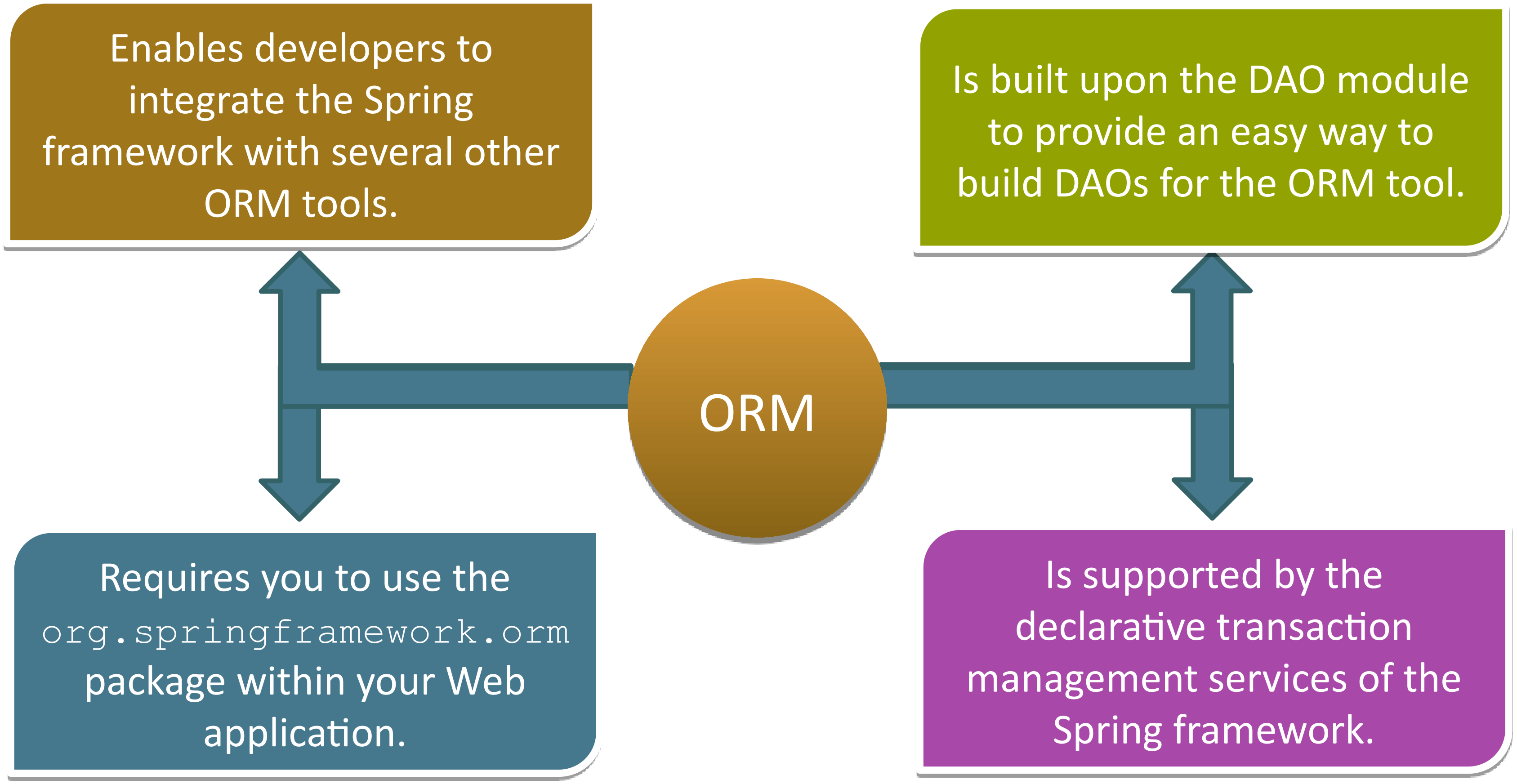
Spring Core

DAO: *The DAO package provides a JDBC-abstraction layer that removes the need to do tedious JDBC coding and parsing of database-vendor specific error codes.*

The JDBC package provides a way to do programmatic as well as declarative transaction management, not only for classes implementing special interfaces, but for all your POJOs (plain old Java objects).

ORM: *The ORM package provides integration layers for popular object-relational mapping APIs, including JPA, JDO, Hibernate, and iBatis. Using the ORM package you can use all those O/R-mappers in combination with all the other features Spring offers, such as the simple declarative transaction management feature mentioned previously.*

ORM



Enables developers to integrate the Spring framework with several other ORM tools.

Is built upon the DAO module to provide an easy way to build DAOs for the ORM tool.

ORM

Requires you to use the `org.springframework.orm` package within your Web application.

Is supported by the declarative transaction management services of the Spring framework.

Spring Core

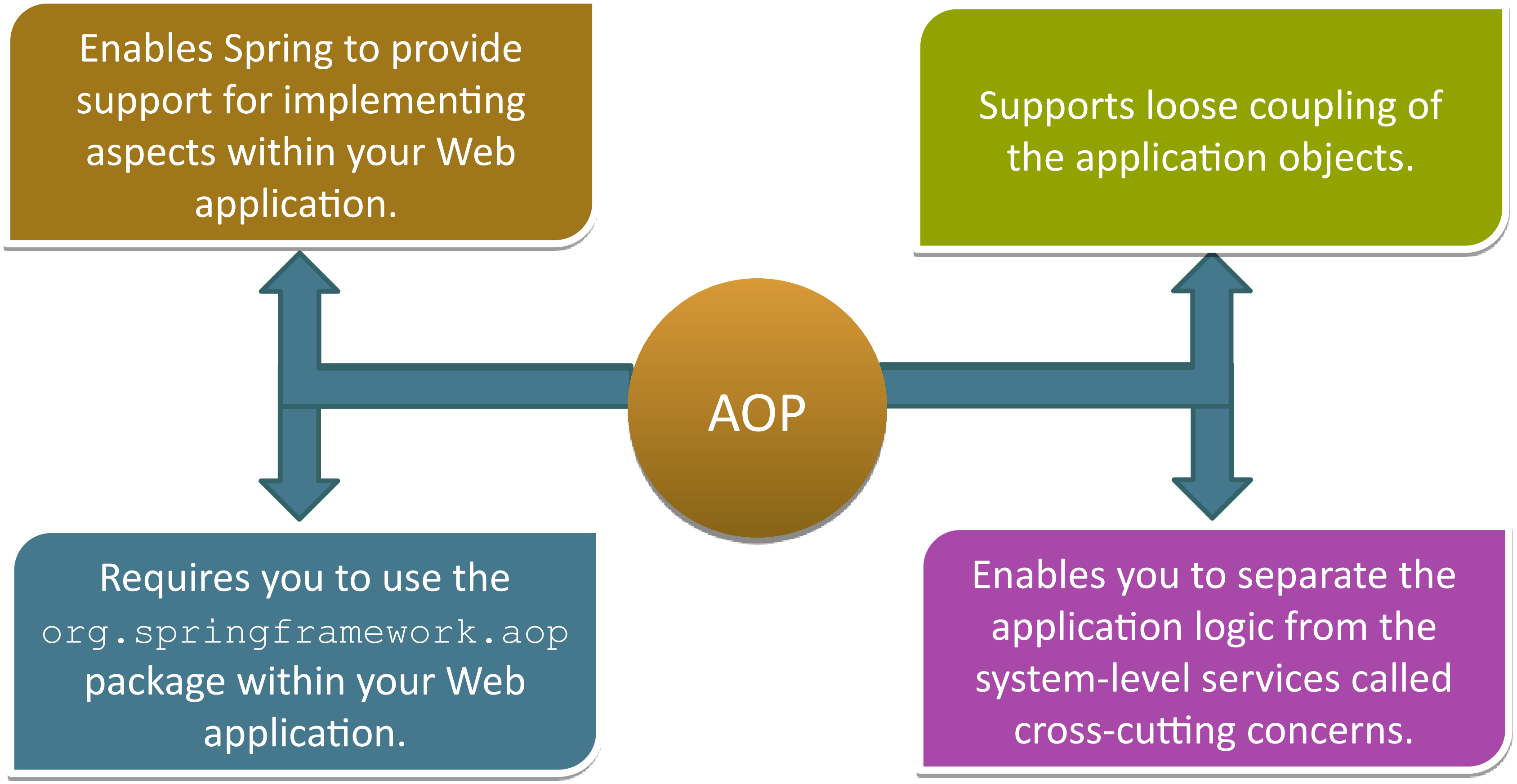
***AOP:** Spring's AOP package provides an AOP Alliance-compliant aspect-oriented programming implementation allowing you to define, for example, method-interceptors and pointcuts to cleanly decouple code implementing functionality that should logically speaking be separated.*

Using source-level metadata functionality you can also incorporate all kinds of behavioral information into your code

***Spring Web:** Spring's Web package provides basic web-oriented integration features, such as multipart file-upload functionality, the initialization of the IoC container using servlet listeners and a web-oriented application context.*

When using Spring together with WebWork or Struts, this is the package to integrate with.

AOP


Enables Spring to provide support for implementing aspects within your Web application.
Supports loose coupling of the application objects.
AOP
Requires you to use the org.springframework.aop package within your Web application.
Enables you to separate the application logic from the system-level services called cross-cutting concerns.

Spring MVC: *Spring's MVC package provides a Model-View-Controller (MVC) implementation for web-applications.*

Spring's MVC framework is not just any old implementation; it provides a clean separation between domain

model code and web forms, and allows you to use all the other features of the Spring Framework.

Tight Coupling vs Loose Coupling

Tight Coupling

```
public class Employee {
    Address address;
    Employee(){
    address=new Address();
    }
}
```

Loose Coupling

```
public class Employee {
Address address;
Employee(Address address){
this.address=address;
}
}
```

MVC

Provides a flexible Model -View-Controller implementation for creating Web applications.

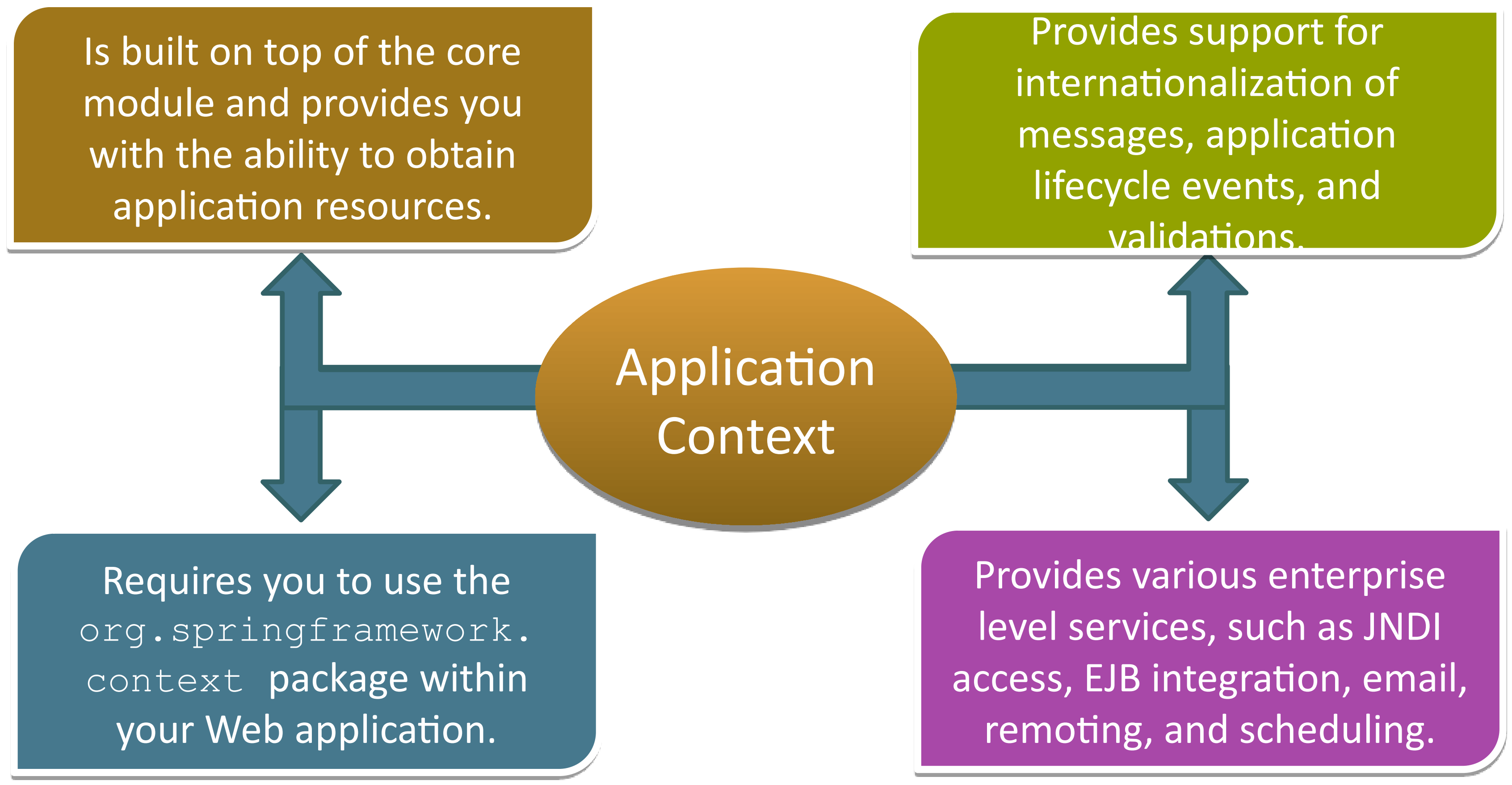
Enables you to separate the model and application logic from the view or the application UI.

MVC

Requires you to use the `org.springframework.web` package within your Web application.

Enables you to integrate several other MVC frameworks, such as Struts, and WebWorks.

Application Context



Application Context

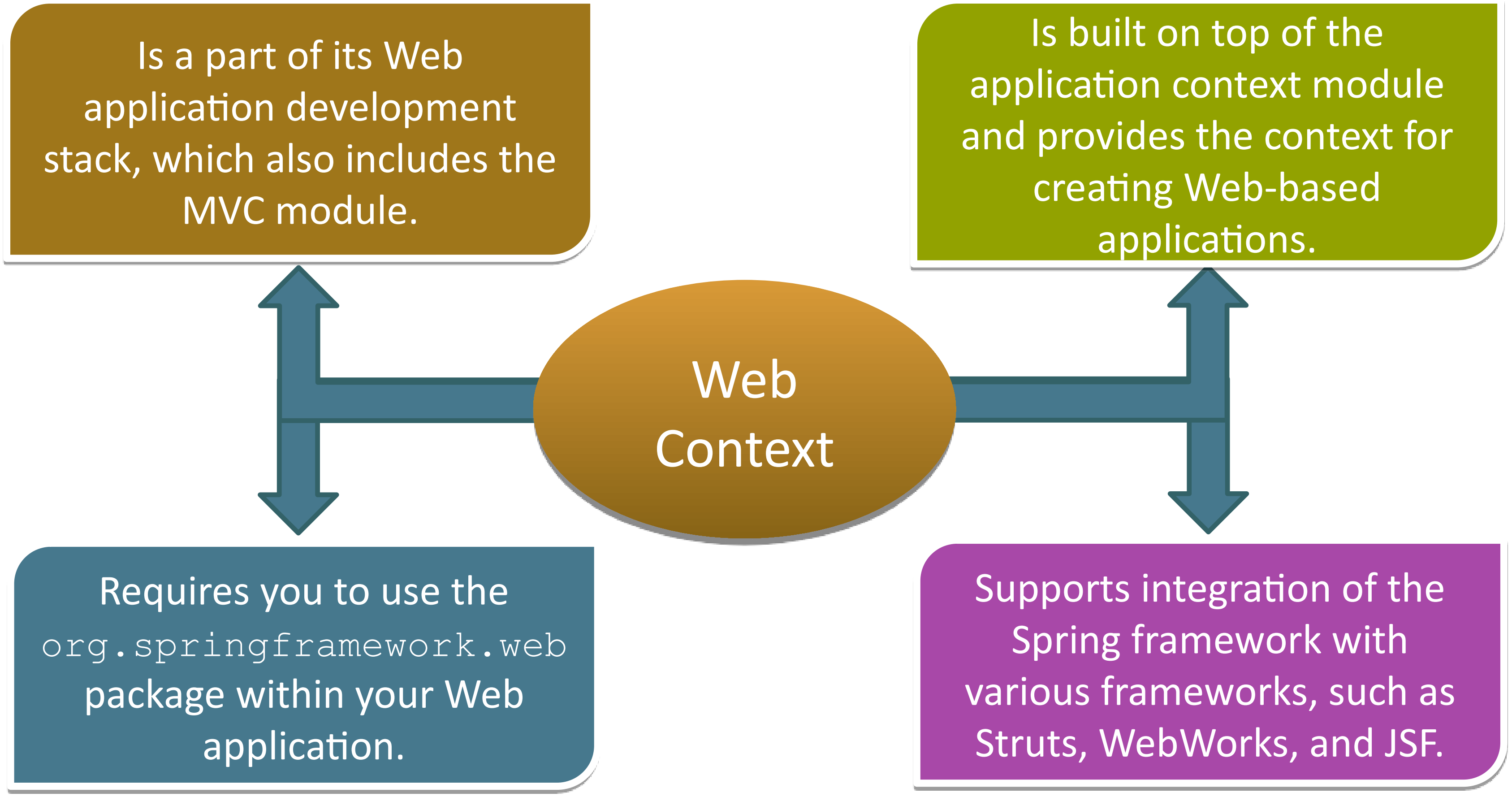
Is built on top of the core module and provides you with the ability to obtain application resources.

Provides support for internationalization of messages, application lifecycle events, and validations.

Requires you to use the `org.springframework.context` package within your Web application.

Provides various enterprise level services, such as JNDI access, EJB integration, email, remoting, and scheduling.

Web Context



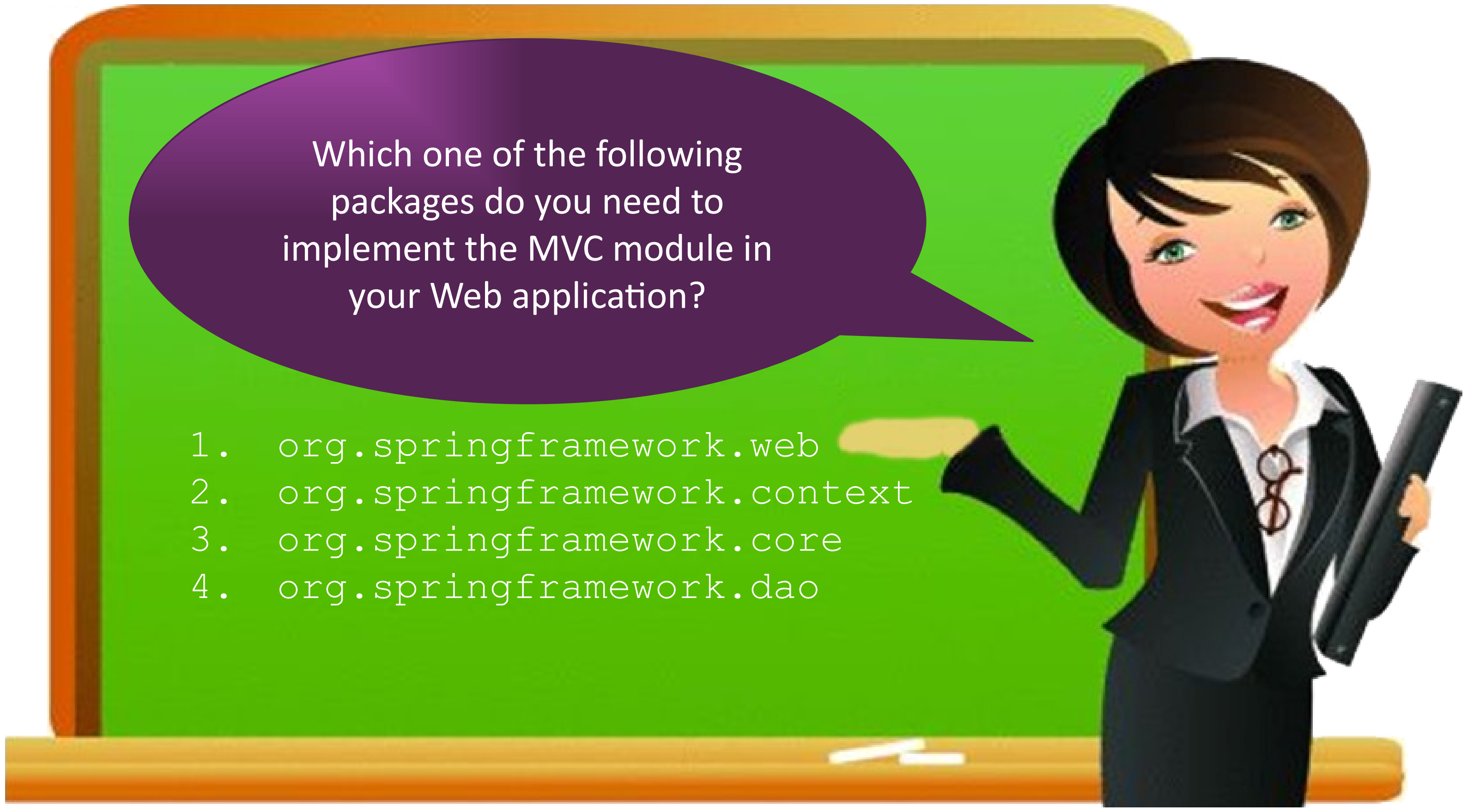
Is a part of its Web application development stack, which also includes the MVC module.

Is built on top of the application context module and provides the context for creating Web-based applications.

Requires you to use the `org.springframework.web` package within your Web application.

Supports integration of the Spring framework with various frameworks, such as Struts, WebWorks, and JSF.

Just a Minute

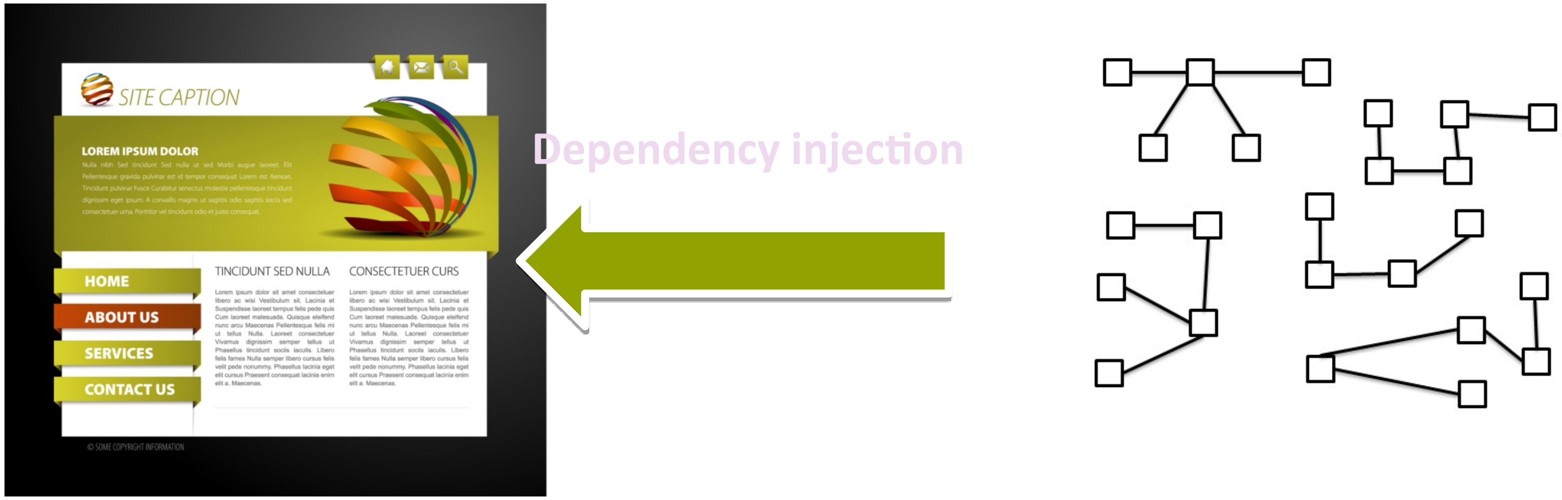


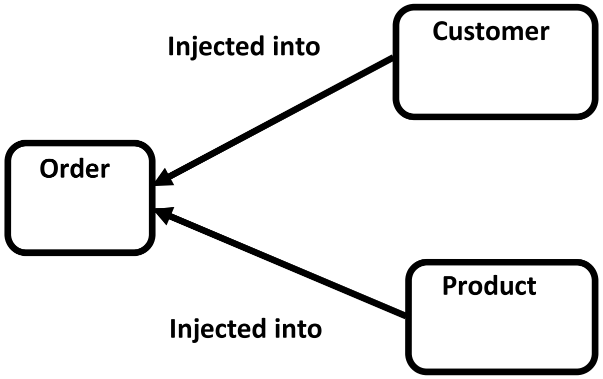
Which one of the following packages do you need to implement the MVC module in your Web application?

1. `org.springframework.web`
2. `org.springframework.context`
3. `org.springframework.core`
4. `org.springframework.dao`

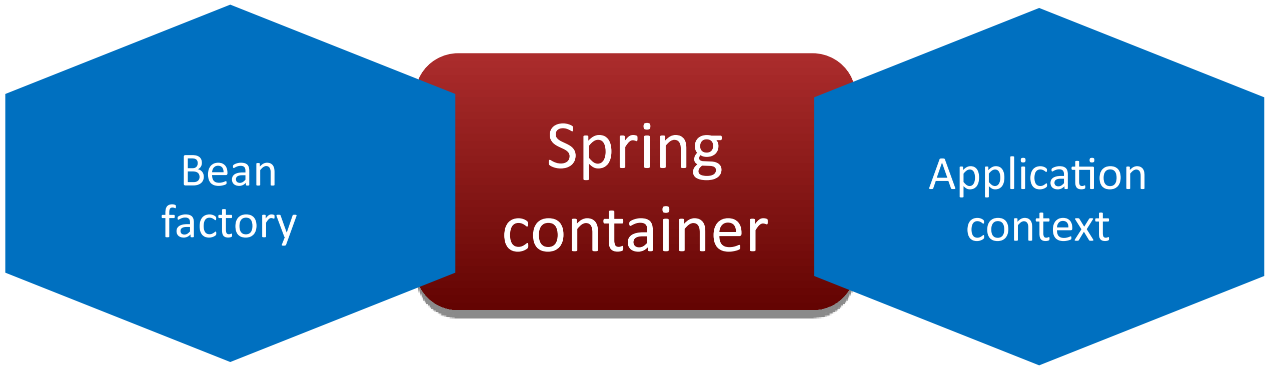
Answer: 1. `org.springframework.web`

Managing Application Objects

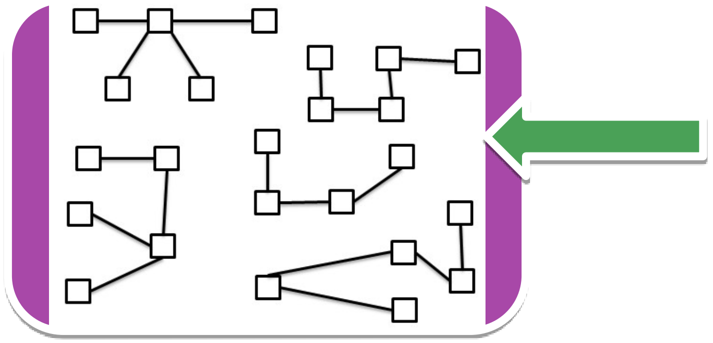




Managing Application Objects (Contd.)



Introducing Bean Factory



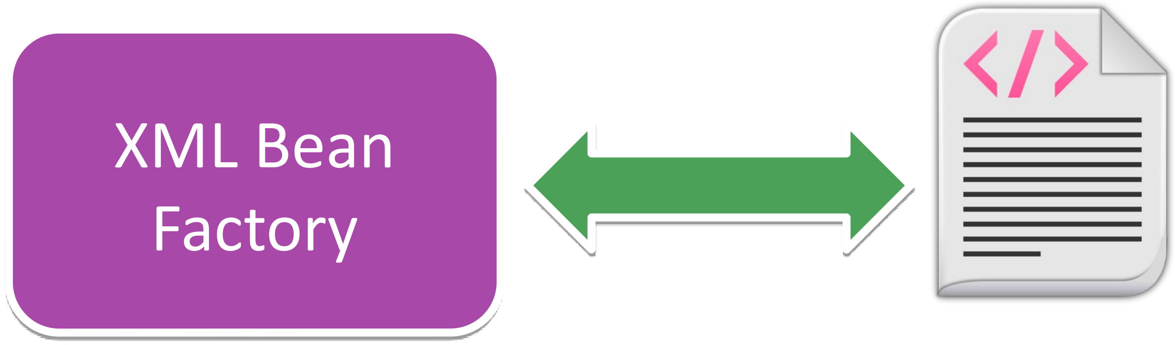
- *The objects created by the bean factory are fully configured, ready to use, and aware of their relationships with the other application objects.*

- *The bean factory is involved in managing the life cycle of objects that have been created.*

- *A bean factory is represented by the* `org.springframework.beans.factory.BeanFactory` *interface.*

- *The most commonly used bean factory is the XML bean factory represented by the* `org.springframework.beans.factory.xml.XmlBeanFactory` *interface.*

Introducing Bean Factory (Contd.)



Syntax of a bean configuration file:

```
<?xml version="1.0" 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/bean
s http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd
">
   ...
</beans>
```

Introducing Bean Factory (Contd.)

- ***In a bean configuration file, each bean is defined by using the <bean> tag within the <beans> tag.***

- ***A bean definition contains the basic information about a bean that a container must know, such as the process for creating a bean, details about the bean life cycle, and the various dependencies for that bean.***

Attributes of the `<beans>` tag

`id`

Used to uniquely identify a bean in the Spring container.

`name`

Used to specify an alias name for the bean.

`class`

Used to specify the fully-qualified (package name + class name) name of the bean class.

`scope`

Used to define the scope of the bean being defined.

Introducing Bean Factory (Contd.)

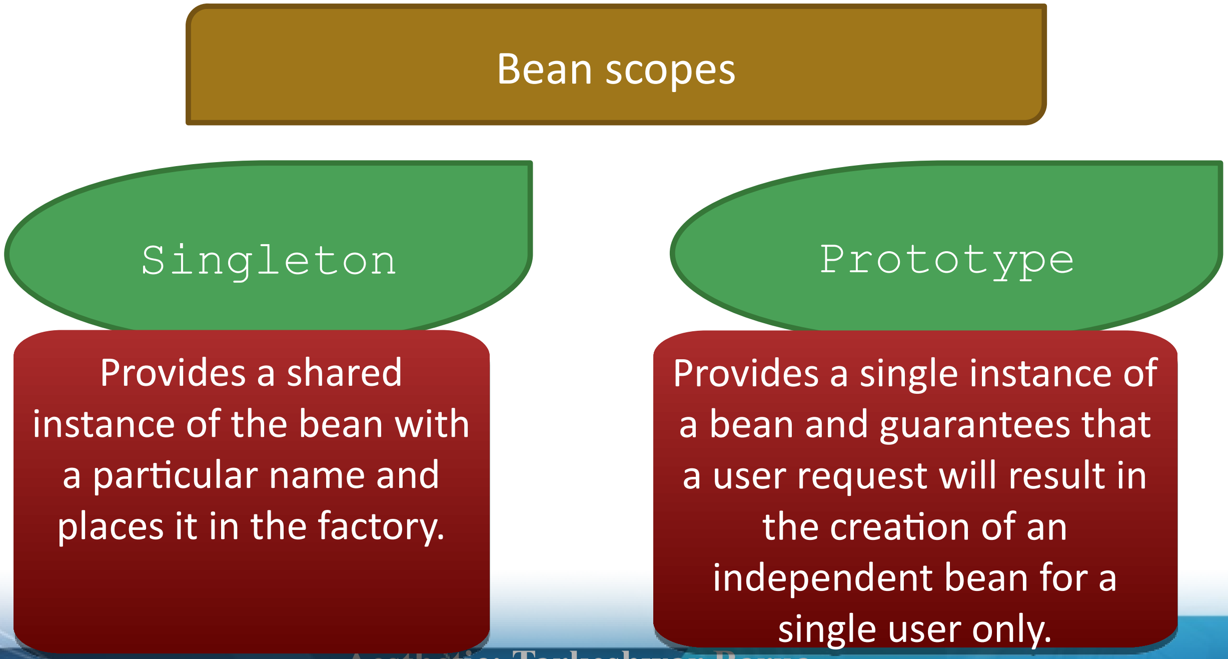
Example: Bean declaration

```
<?xml version="1.0" 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/bean
s http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd
">
<bean id="Beckham" class="Sport.LAGalaxy"/>
</beans>
```

Introducing Bean Factory (Contd.)

- *To instantiate a bean factory in your Web application, you first need to load the bean configuration file.*
- *This can be done by using the `Resource` object, defined by the* `org.springframework.core.io.Resource` *interface.*
- *Depending upon the place from where the bean configuration file is to be loaded, the Spring framework provides several implementations of the `Resource` interface.*



Provides a shared instance of the bean with a particular name and places it in the factory.

Provides a single instance of a bean and guarantees that a user request will result in the creation of an independent bean for a single user only.

Introducing Bean Factory (Contd.)

Bean instantiation

- ***You can retrieve an instance of a bean registered under a given name from the bean factory by using the `getBean()` method.***
- ***This method takes the ID of the bean as a string value.***

Example:

```
LAGalaxy team = (LAGalaxy)
factory.getBean("Beckham");
team.squadSize();
```

- ***When the `getBean()` method is called, the bean factory container instantiates the bean and set its properties through DI.***
- ***After this, the life of the bean begins inside the Spring container.***

Introducing Bean Factory (Contd.)

```
Resource resource=new
ClassPathResource("applicationContext.xml");
BeanFactory factory=new XmlBeanFactory(resource);

Student student=(Student)factory.getBean("studentbean");
```

Bean
instantiation

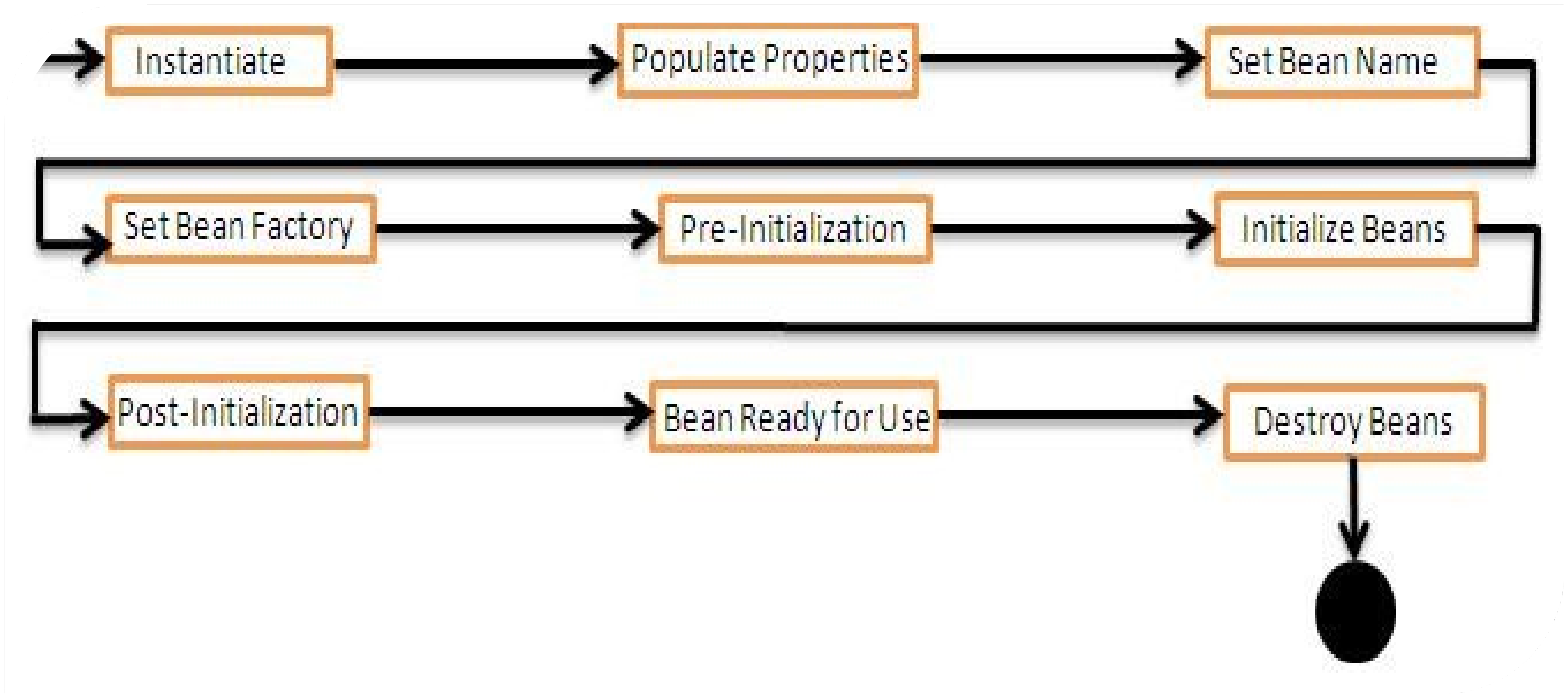
JNDI lookup:

```
Context ctx = new InitialContext();
Context environmentCtx = (Context)
ctx.lookup("java:comp/env");
A obj = (A)environmentCtx.lookup("A");
```

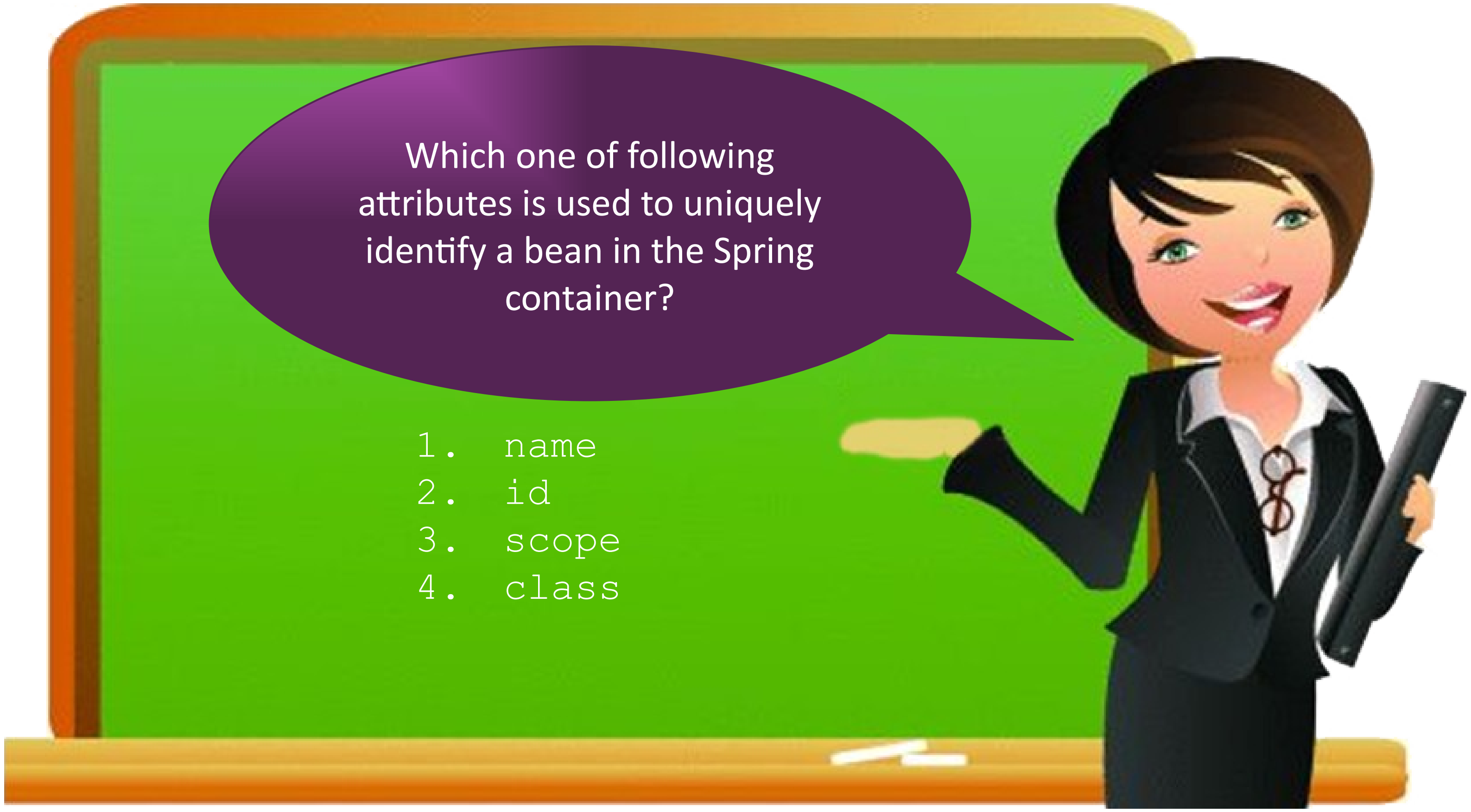
```
<constructor-arg value="10" type="int"></constructor-arg>
```

Introducing Bean Factory (Contd.)

Life cycle of a bean inside the bean factory



Just a Minute



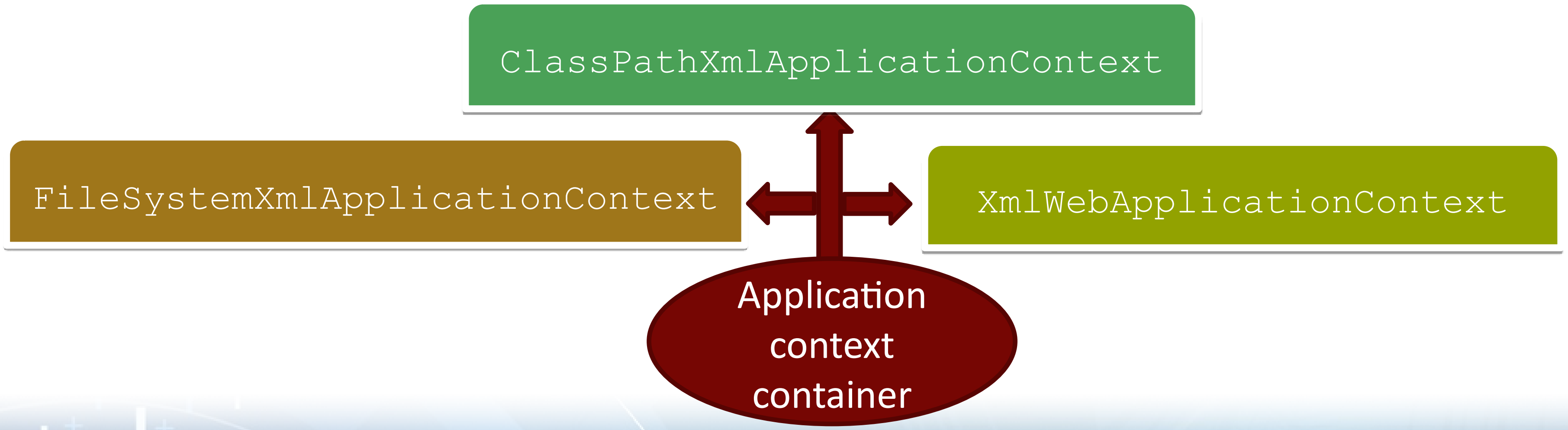
Which one of following attributes is used to uniquely identify a bean in the Spring container?

```
1.  name
2.  id
3.  scope
4.  class
```

Answer: `2. id`

Introducing Application Context

- ***Provides a means for resolving text messages, including support for internationalization of those messages.***
- ***Provides a generic way to load file resources, such as images.***
- ***Publishes events to beans, which are registered as listeners.***


ClassPathXmlApplicationContext
FileSystemXmlApplicationContext
XmlWebApplicationContext
Application context container

Introducing Application Context (Contd.)

Loading the application context

- ***If you want to use the application context container for managing your application objects, then you have to load a particular `ApplicationContext` implementation.***

Example: `ClassPathXmlApplicationContext`

```
ApplicationContext appContext = new
ClassPathXmlApplicationContext("Sport/Spring-
Config.xml");
```

Example: `FileSystemXmlApplicationContext`

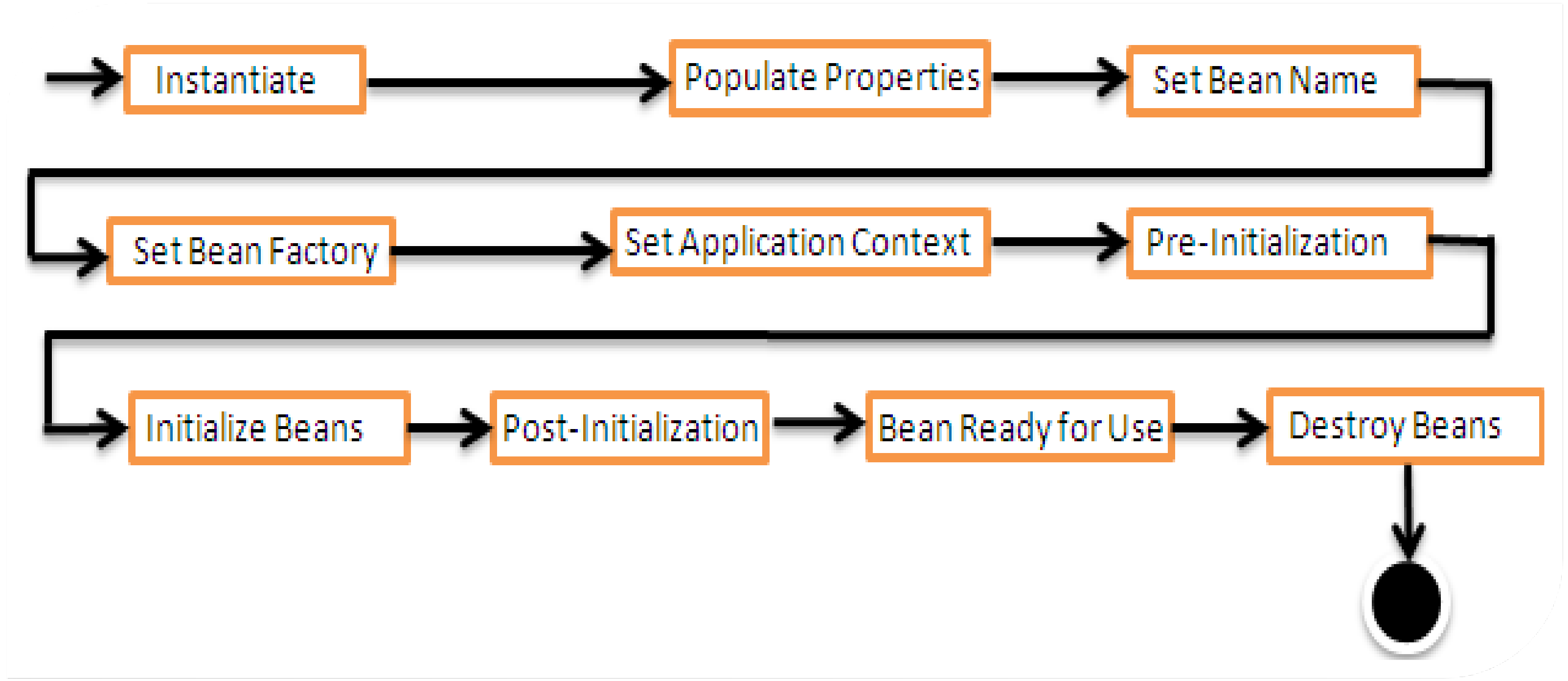
```
ApplicationContext appContext = new
FileSystemXmlApplicationContext("c:/Sport/Spr
ing-Config.xml");
```

Example: `XmlWebApplicationContext`

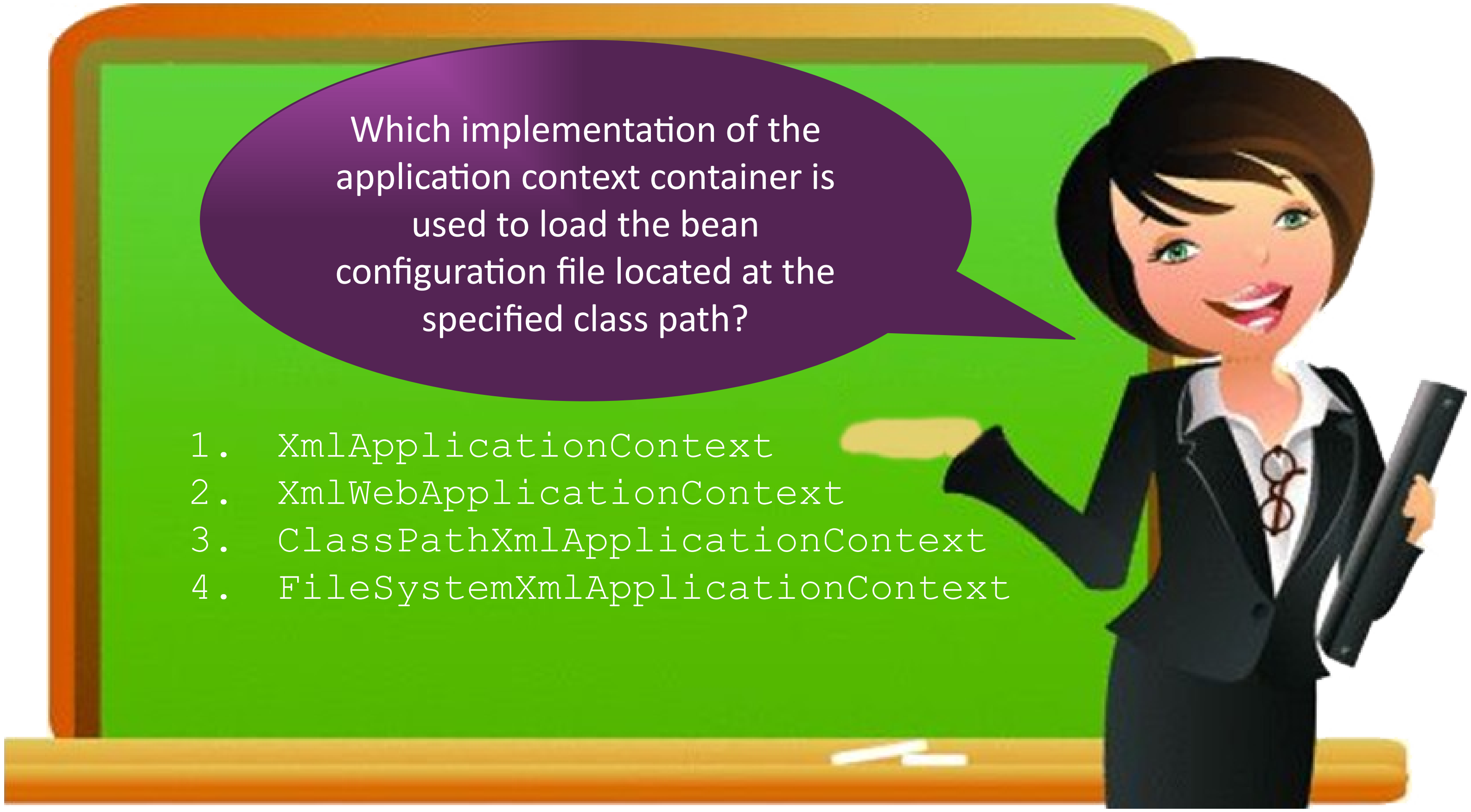
```
XmlWebApplicationContext appContext = new
XmlWebApplicationContext();
```

Introducing Application Context (Contd.)

Life cycle of a bean inside application context



Just a Minute



Which implementation of the application context container is used to load the bean configuration file located at the specified class path?

1. `XmlApplicationContext`
2. `XmlWebApplicationContext`
3. `ClassPathXmlApplicationContext`
4. `FileSystemXmlApplicationContext`

Answer: 3. `ClassPathXmlApplicationContext`

Autowiring in Spring

- Autowiring feature of spring framework enables you to inject the object dependency implicitly. It internally uses setter or constructor injection
- Autowiring can't be used to inject primitive and string values. It works with reference only

Mode	**Description**
byName	The byName mode injects the object dependency according to name of the bean. In such case, property name and bean name must be same. It internally calls setter method.
byType	The byType mode injects the object dependency according to type. So property name and bean name can be different. It internally calls setter method.
constructor	The constructor mode injects the dependency by calling the constructor of the class. It calls the constructor having large number of parameters.
No	It is the default autowiring mode. It means no autowiring bydefault.
autodetect	deprecated since Spring 3

Autowiring in Spring

- <bean id="person" class="org.tarkesh.Person" autowire="byName"></bean>

```
package org.tarkesh;
public class Person {
Person(){System.out.println("person is created");}
void print(){System.out.println("hello Person");}
}
```

```
package org.tarkesh;
public class Laptop {
Person person;
Person(){System.out.println("Person is created");}
public Person getPerson() {
    return laptop;
}
public void setPerson(Person person) {
    this.person = person;
}
void print(){System.out.println("hello Laptop");}
void display(){
    print();
    person.print();  }  }
```

Autowiring byType in Spring

```
<bean id="person" class="org.tarkesh.Person"></bean>
<bean id="laptop" class="org.tarkesh.Laptop" autowire="byType"></bean>
```

In case of byType autowiring mode, bean id and reference name may be different. But there must be only one bean of a type.

Autowiring constructor in Spring

```
<bean id="person" class="org.tarkesh.Person"></bean>
<bean id="laptop" class="org.tarkesh.Laptop" autowire="constructor"></bean>
```

In case of constructor autowiring mode, spring container injects the dependency by highest parameterized constructor.

If you have 3 constructors in a class, zero-arg, one-arg and two-arg then injection will be performed by calling the two-arg constructor.

Autowiring constructor in Spring

```
<bean id="person" class="org.tarkesh.Person"></bean>
<bean id="laptop" class="org.tarkesh.Laptop" autowire="no"></bean>
```

In case of no autowiring mode, spring container doesn't inject the dependency by autowiring.

List in constructor injection

```
<constructor-arg>
<list>
<value>Java is a Programming Language</value>
<value>Java is a Platform</value>
<value>Java is an Island</value>
</list>
</constructor-arg>
```

```
public class Question {
    private int id;
    private String name;
    private List<String> answers;

    public Question(int id, String name, List<String> answers)
{
        super();
        this.id = id;
        this.name = name;
        this.answers = answers;
    }
}
```

Set in constructor injection

```
<constructor-arg>
<set>
<value>Java is a Programming Language</value>
<value>Java is a Platform</value>
<value>Java is an Island</value>
</set>
</constructor-arg>
```

```
public class Question {
    private int id;
    private String name;
    private Set<String> answers;

    public Question(int id, String name, Set<String> answers)
{
        super();
        this.id = id;
        this.name = name;
        this.answers = answers;
    }
}
```

Map in constructor injection

```
<constructor-arg>
<map>
<entry key="Java is a Programming Language"  value="Mr. ABC"></entry>
<entry key="Java is a Platform" value="Mr. KBC"></entry>
<entry key="Java is an Island" value="Mr. XYZ"></entry>
</map>
</constructor-arg>
```

```
public class Question {
    private int id;
    private String name;
    private Map<String, String> answers;

    public Question(int id, String name, Map<String, String>
answers) {
        super();
        this.id = id;
        this.name = name;
        this.answers = answers;
    }
}
```

Set in constructor injection

```
<constructor-arg>
<list>
<ref bean="ans1"/>
<ref bean="ans2"/>
</list>
</constructor-arg>
```

```
public class Question {
    private int id;
    private String name;
    private List<String> answers;

    public Question(int id, String name, List<String> answers)
{
        super();
        this.id = id;
        this.name = name;
        this.answers = answers;
    }
}
```

Set in constructor injection

```
<constructor-arg>
<map>
<entry key-ref="answer1" value-ref="user1"></entry>
<entry key-ref="answer2" value-ref="user2"></entry>
</map>
</constructor-arg>
```

```
public class Question {
    private int id;
    private String name;
    private List<String> answers;

    public Question(int id, String name, List<String> answers)
{
        super();
        this.id = id;
        this.name = name;
        this.answers = answers;
    }
}
```