SPRING FRAMEWORK AND ITS COMPONENTS

SPRING FRAMEWORK

What is spring?

- Open source, light weight solution for building enterprise application
- Developers can make application using BEANS and POJO
- Integrate Spring framework with other existing technologies
- A complete solution to develop an enterprise application

WHY SPRING?

Spring framework is the first choice

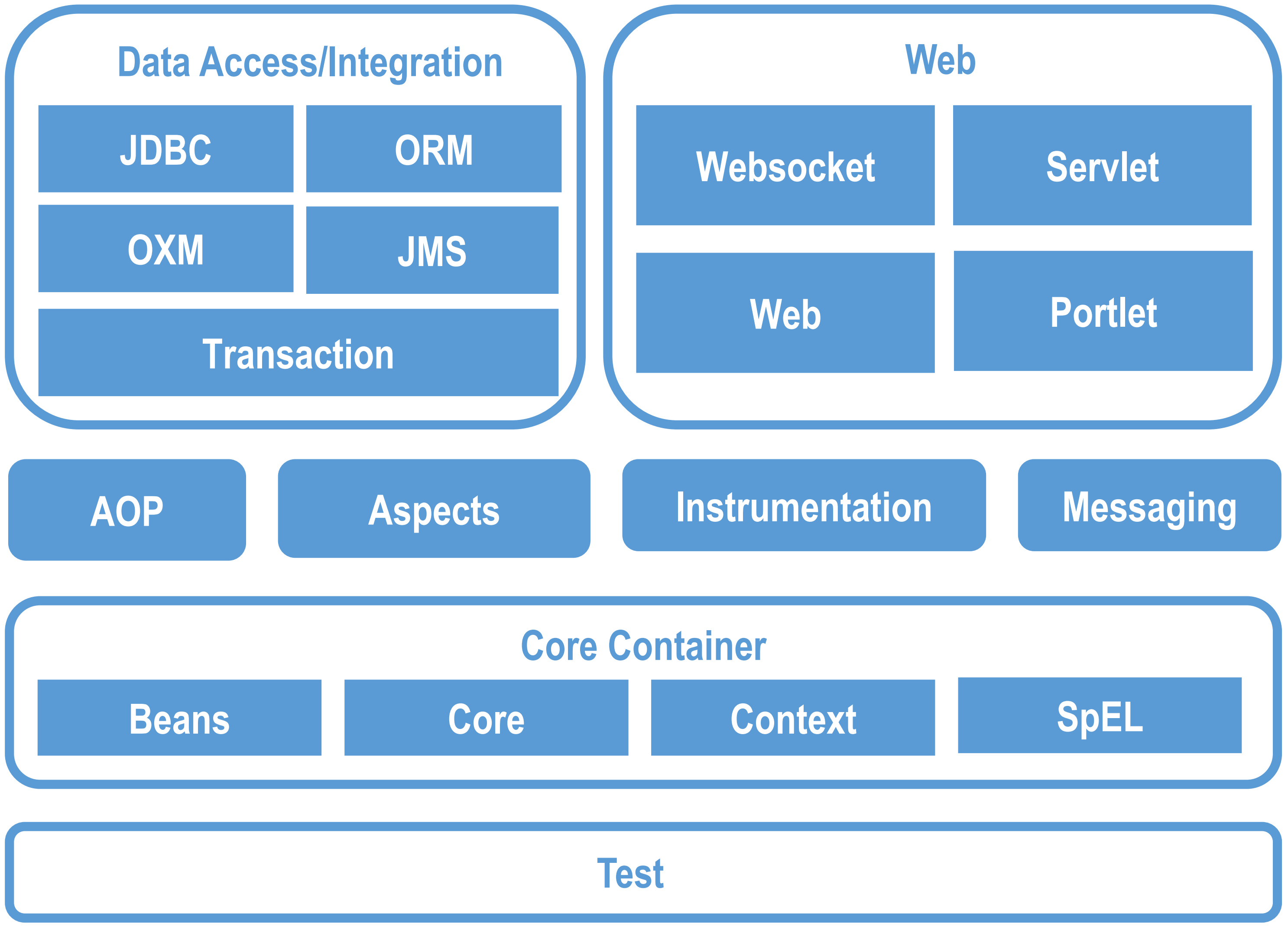
[illegible]

Minimize cost

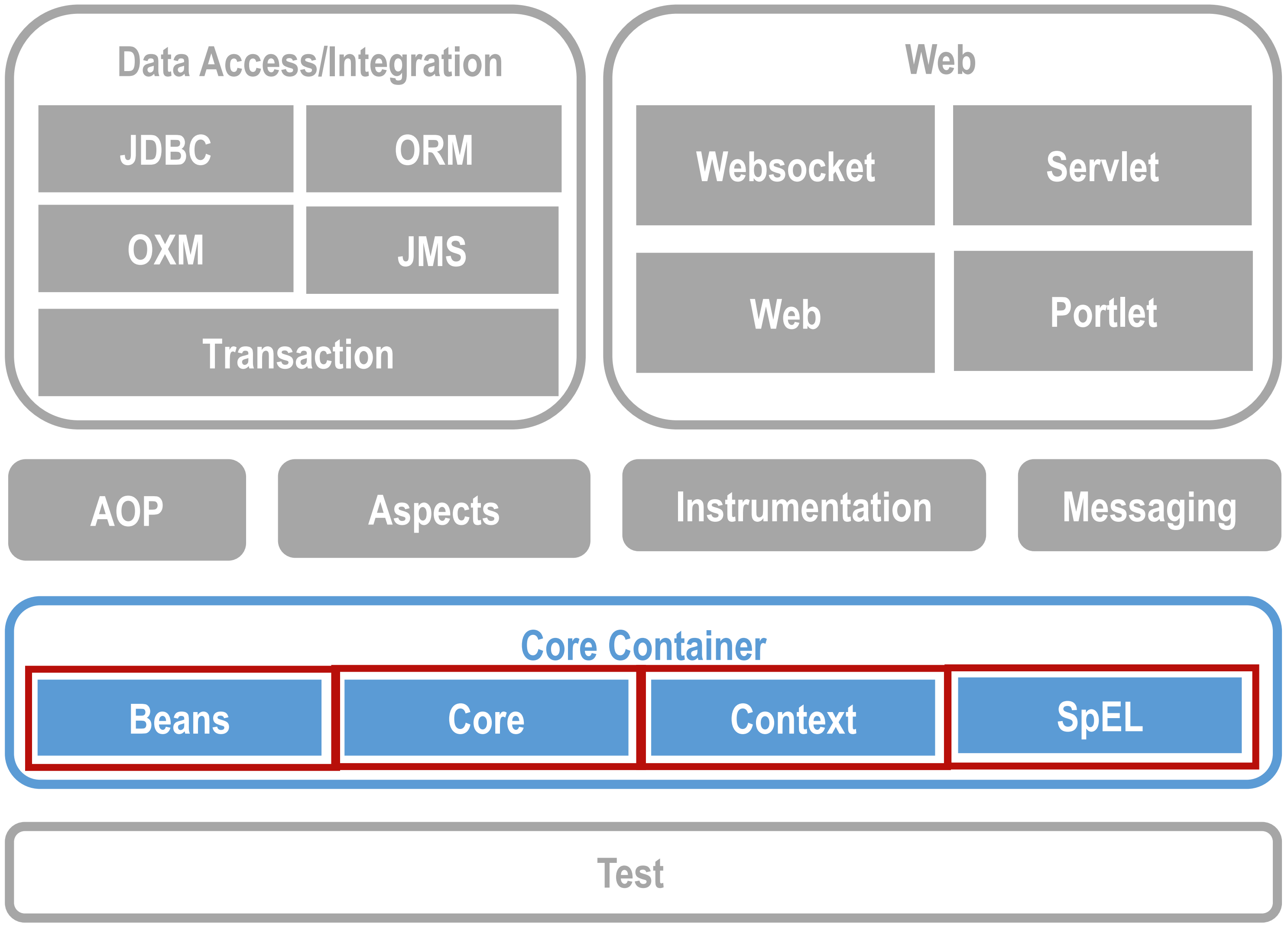
WHY SPRING?

- Minimizes dependency complexities and provides more clean and maintainable code
- Provides declarative programming with AOP
- Reduces repetitive coding
- Provides appropriate templates
- Provides declarative Transaction Management

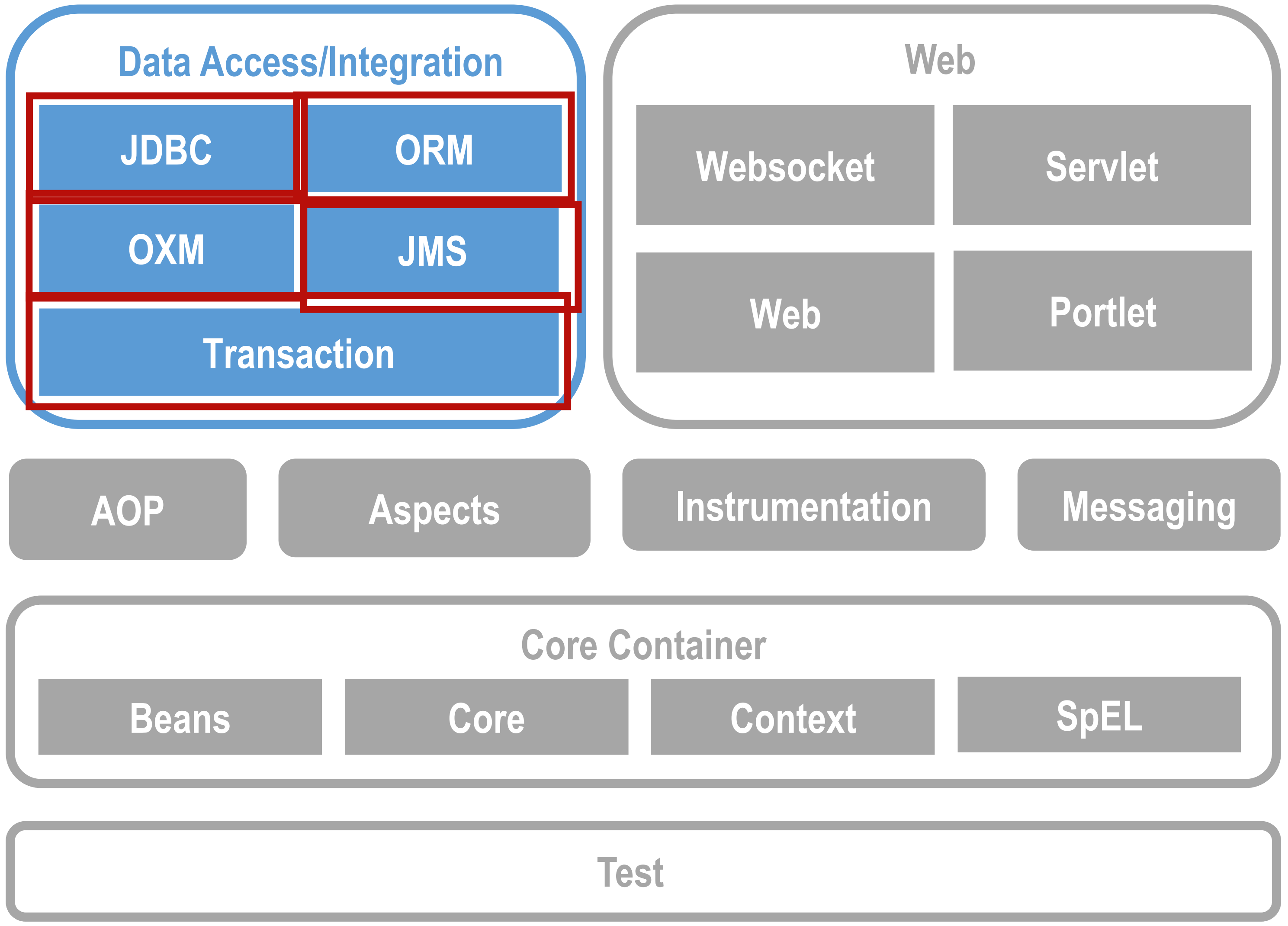
SPRING RUNTIME COMPONENTS



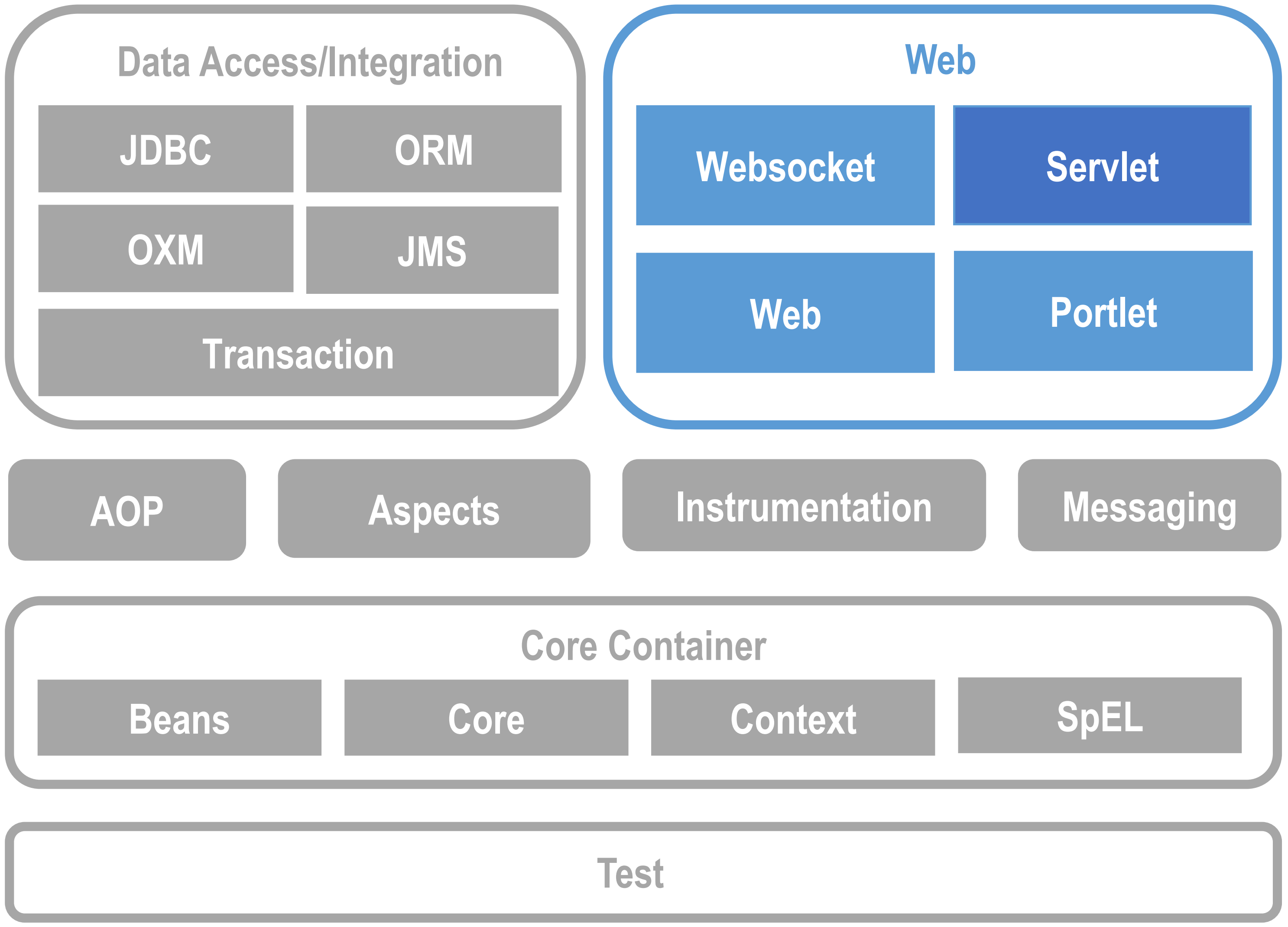
SPRING RUNTIME COMPONENTS - CORE CONTAINER MODULES



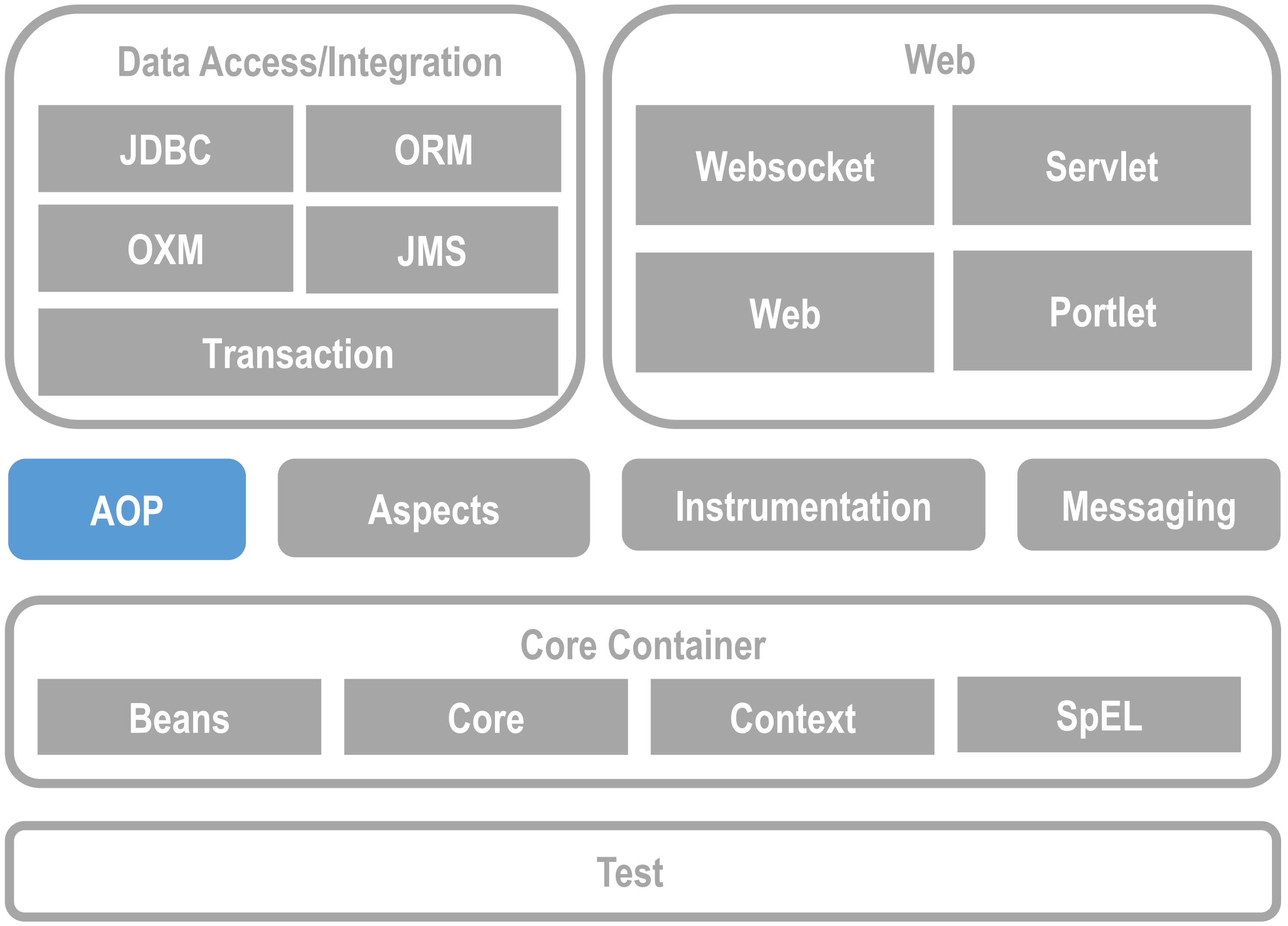
SPRING RUNTIME COMPONENTS - DATA ACCESS/INTEGRATION MODULE



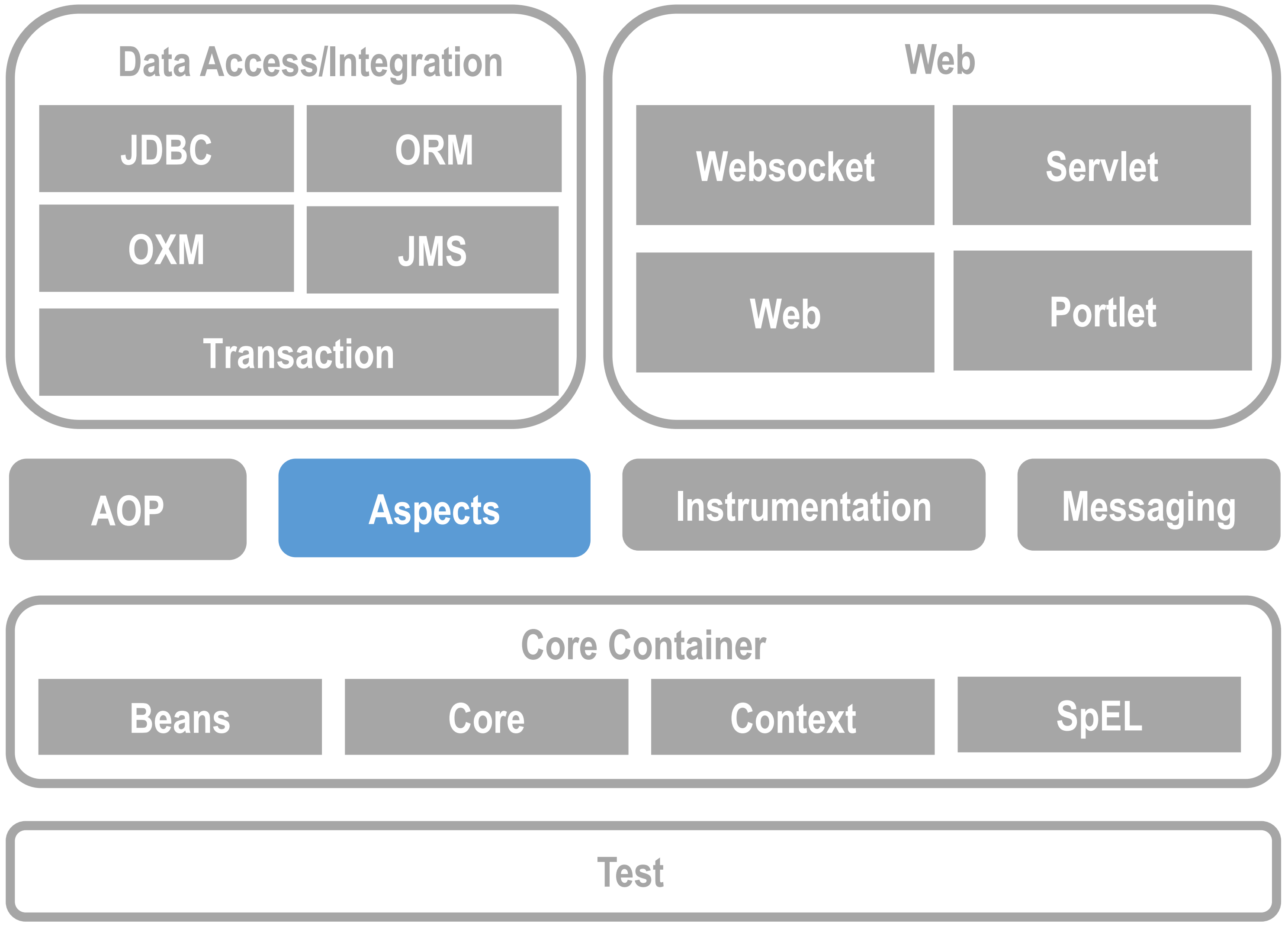
SPRING RUNTIME COMPONENTS - WEB LAYER MODULE



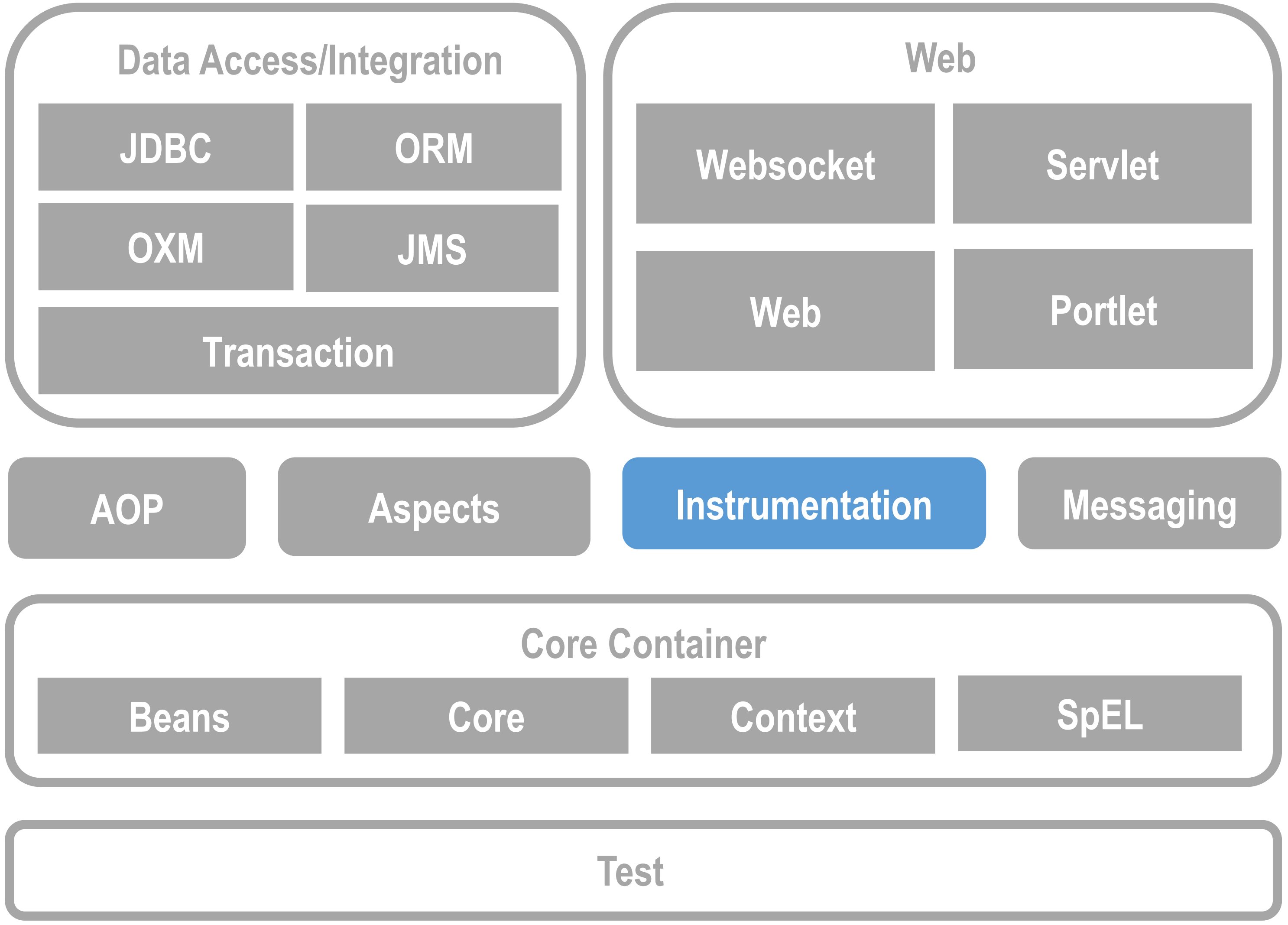
SPRING RUNTIME COMPONENTS - AOP MODULE



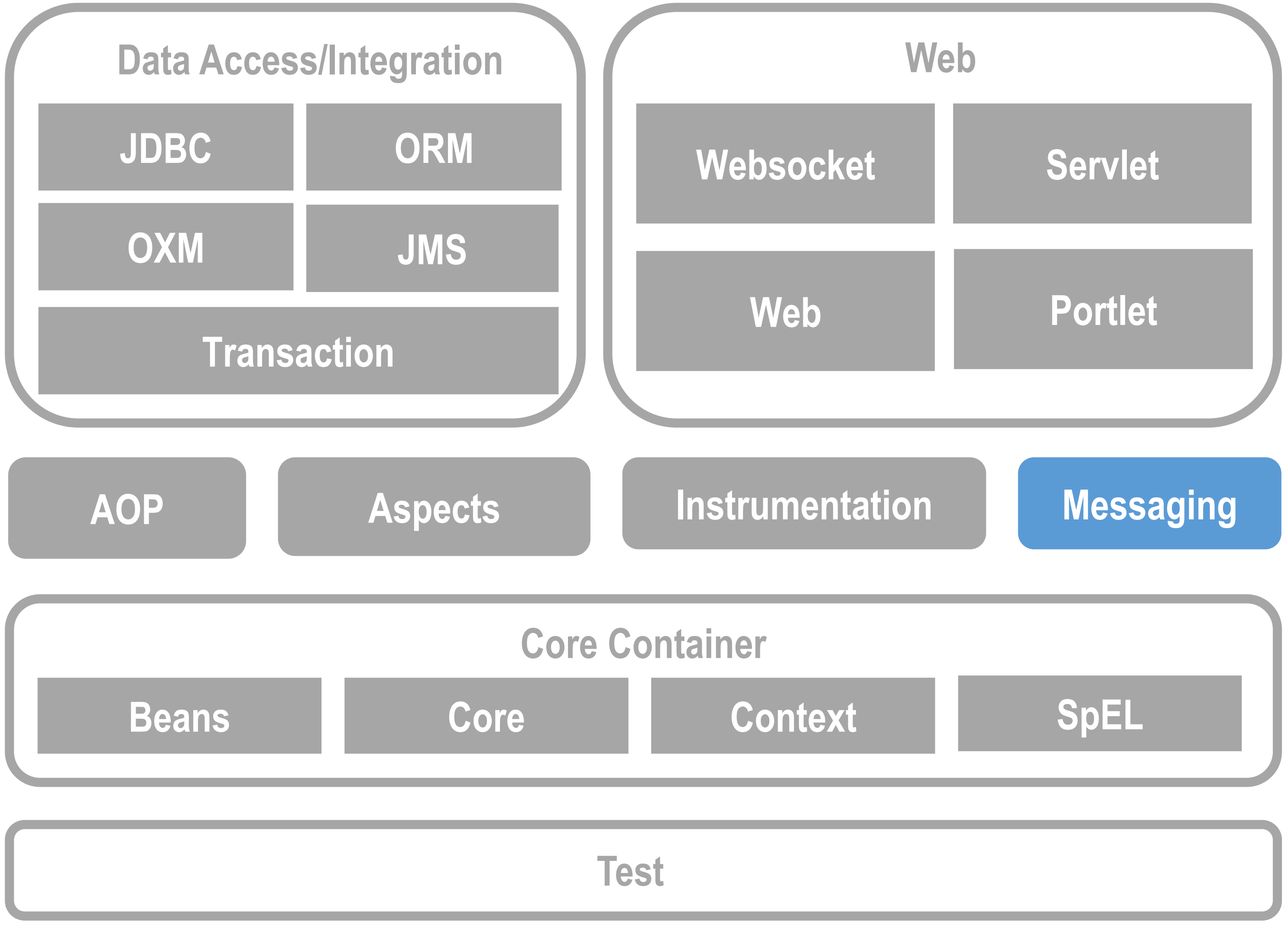
SPRING RUNTIME COMPONENTS - ASPECTS MODULE



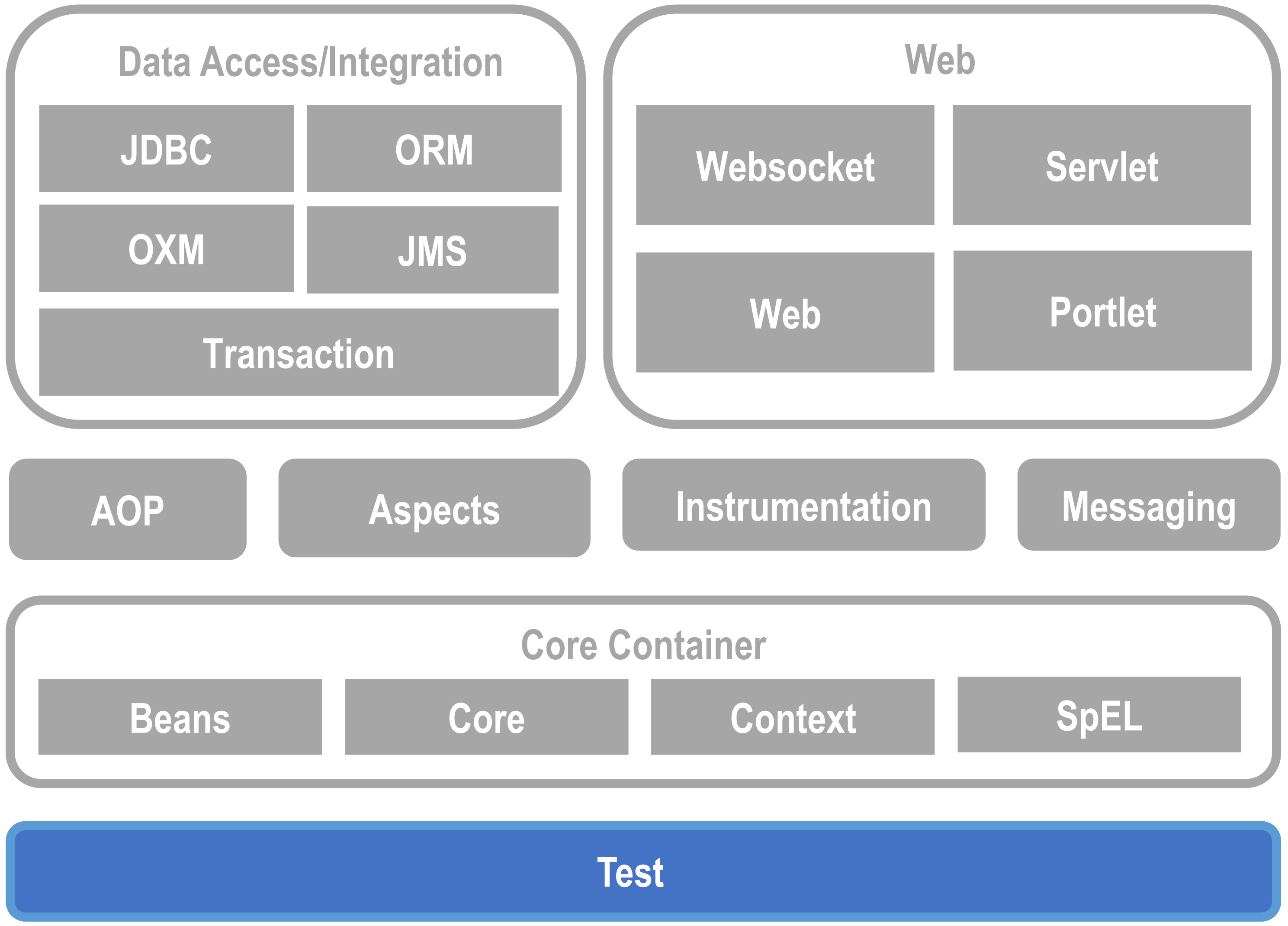
SPRING RUNTIME COMPONENTS - INSTRUMENTATION MODULE



SPRING RUNTIME COMPONENTS - MESSAGING MODULE



SPRING RUNTIME COMPONENTS - TEST MODULE



SPRING TOOL SUITE

IDES FOR BUILDING SPRING APPLICATIONS

Types of IDEs

- Eclipse
- IntelliJ
- NetBeans
- Spring Tool Suite

Additional support to develop Spring Application

SPRING TOOL SUITE

Support to implement, debug, deploy and execute spring applications

Contains developer edition of Pivotal tc Server within it

Dependencies will be added in the pom.xml automatically

SPRING TOOL SUITE – INSTALLATION & CONFIGURATION

jdk is the prerequisite for installing any **IDE** for **java application**

Download JDK1.8

01

Download **JDK1.8** or higher from **http://www.oracle.com** and install it in your system and do the path Setting

Download STS zip file

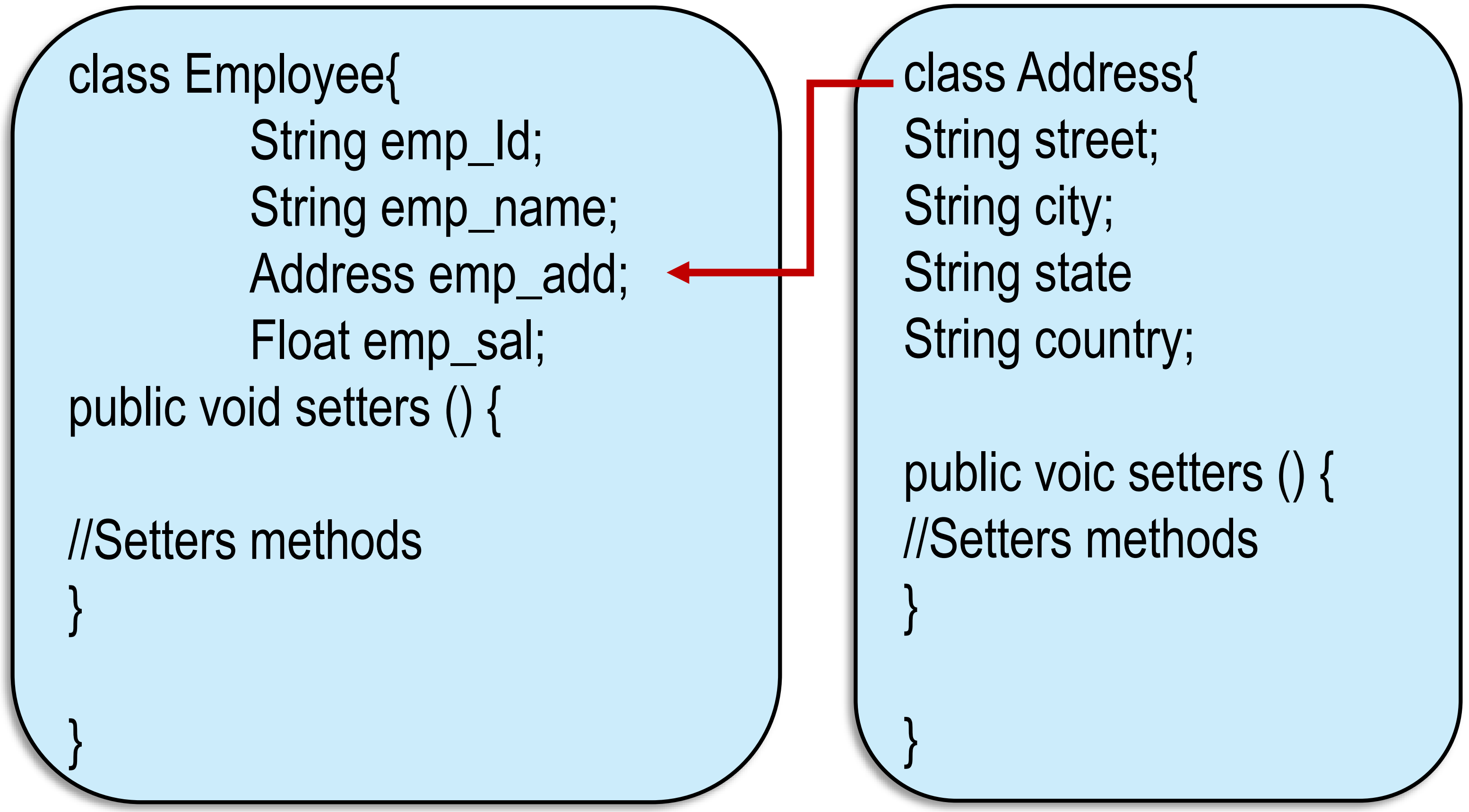
02

Download latest version of **Spring Tool Suite (STS) zip** file from **https://spring.io/tools** and unzip it

WORKING OF SPRING APPLICATION

WORKING OF SPRING APPLICATION

EmployeeBean.java



```
class Employee{
            String emp_Id;
            String emp_name;
            Address emp_add;
            Float emp_sal;
public void setters () {

//Setters methods
}

}
```

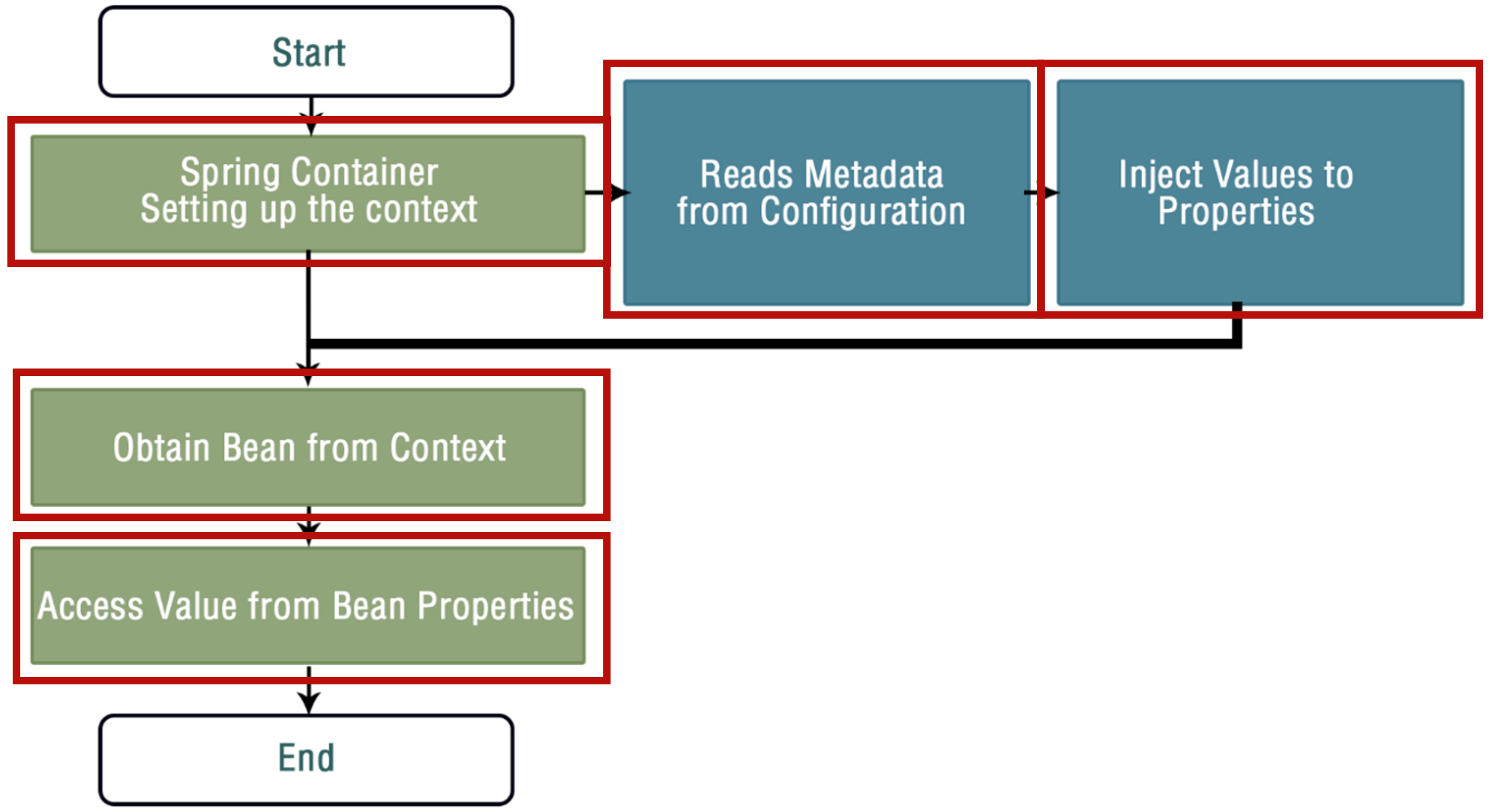
```
class Address{
String street;
String city;
String state
String country;

public voic setters () {
//Setters methods
}

}
```

WORKING OF SPRING APPLICATION

spring.xml

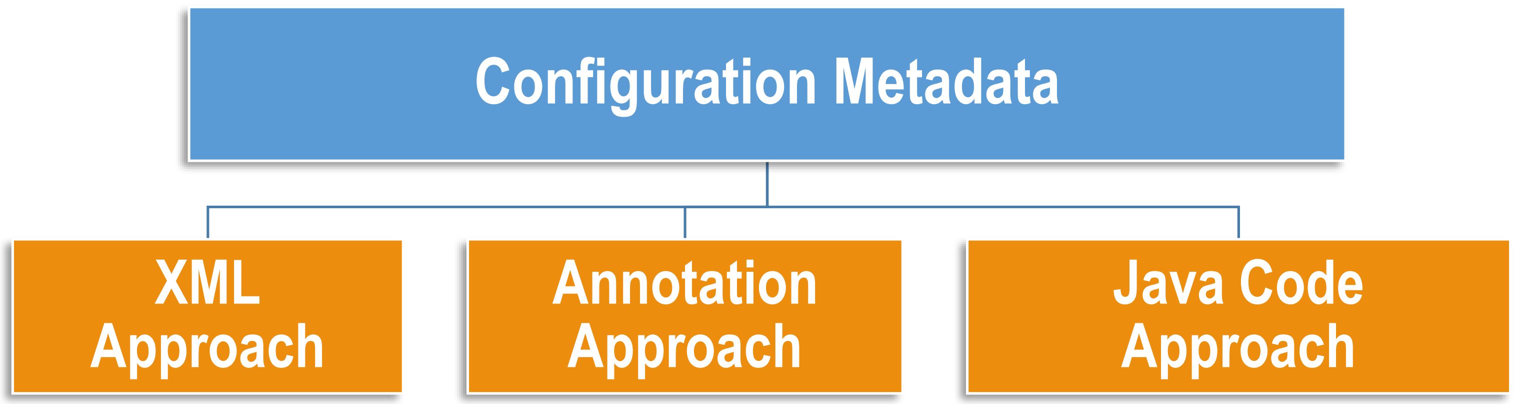
```
<bean id="addr" class="spring.di.Address">
</bean>

<bean id="emp1" class="spring.di.Employee">
            <constructor-arg name="address" ref="addr"/>
</bean>
```

WORKING OF SPRING APPLICATION


Start
Spring Container Setting up the context
Reads Metadata from Configuration
Inject Values to Properties
Obtain Bean from Context
Access Value from Bean Properties
End

ANNOTATION AND JAVA CONFIGURATION

CONFIGURATION METADATA



ANNOTATION BASED CONTAINER CONFIGURATION

Annotation based Configuration

- ✓ Beans can be configured
- ✓ Reduces the XML configuration
- ✓ Use of Component–scan

ANNOTATION BASED CONTAINER CONFIGURATION

Step: 1

```
beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd">

    <context:component-scan base-package="com.spring.bean"/>

 </bean>
</beans>
```

ANNOTATION BASED CONTAINER CONFIGURATION

Step: 2

```
import org.springframework.stereotype.Component;

@Component("empBean")
public class EmployeeBean {

}
```

ANNOTATION BASED CONTAINER CONFIGURATION

Step: 3

```
//getting the Spring Container

ApplicationContext actx= new ClassPathXmlApplicationContext("spring.xml");

//Retrieving the Bean

EmployeeBean emp=
actx.getBean("empBean",EmployeeBean.class);
```

JAVA BASED CONTAINER CONFIGURATION

NO-XML Approach

Will not be writing a single XML file

Annotations in Java configuration

- **@Configuration**
- **@Bean**

JAVA BASED CONTAINER CONFIGURATION

Example of a @Bean method declaration

```
@Configuration
public class AppConfig {

    @Bean
    public BankingService bankingService() {
        return new BankingServiceImpl();
    }

}
```

```
<beans>
<bean
id="bankingService"class="MyServiceImpl"/>
</beans>
```

JAVA BASED CONTAINER CONFIGURATION

01

Bean class

Create a Bean class

02

@Configuration

Create a Configuration class

03

@Bean

Write a method to return the object of a class

04

getBean Method

Create a class with main method

SPRING BEAN

SPRING BEAN

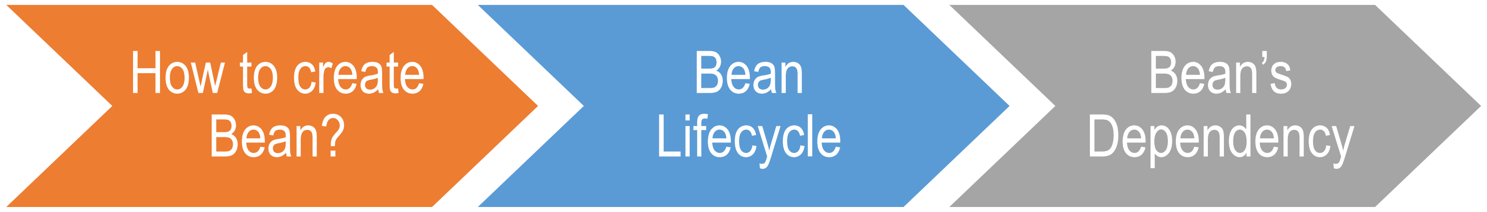
Spring Beans

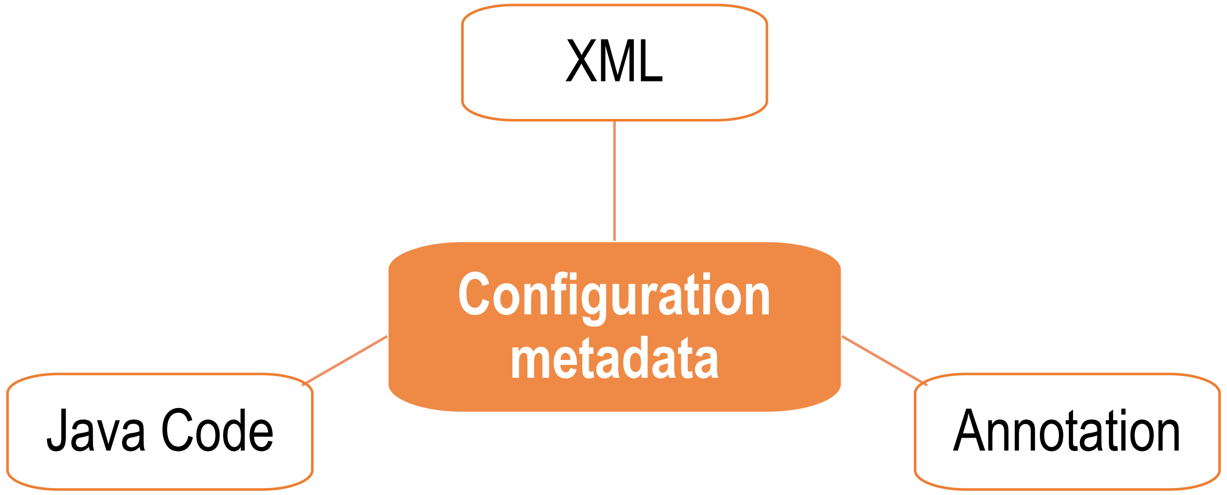
- Basic building blocks of spring framework
- Managed by **the** spring IOC container
- Created with the configuration metadata

Declare beans using <bean> tag in XML

BEAN DEFINITION

Bean definition should contain information such as:





JAVA BASED CONTAINER CONFIGURATION

@Bean

- Method-level annotation
- It can be applied over methods

JAVA BASED CONFIGURATION

@Bean Method Declaration:

```
@Configuration
public class AppConfig {

  @Bean
  public BankService bankingService() {
      return new BankingServiceImpl();
  }

}
```

SPRING BEAN : IMPORTANT POINTS

Container will contain beans as long as they are required by an Application.

Beans created outside Spring container can also be registered with Application Context.

BeanFactory is an interface to accessing the bean container.

Every Bean has lifecycle interface and methods.

LOOSE COUPLING AND INTERFACE

COUPLING

Coupling

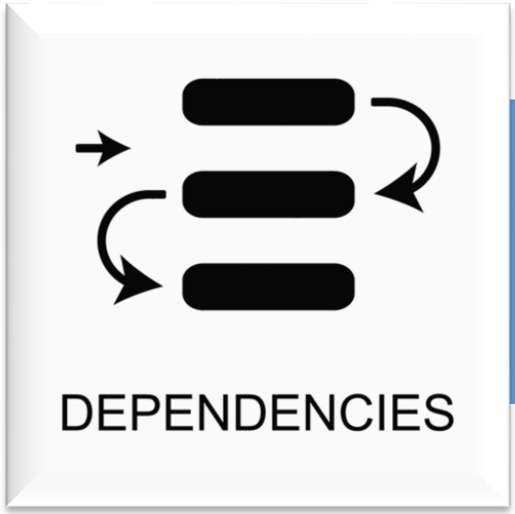
How much one class is dependent on another class

How much a change in a class will force to do the related changes in other class

Spring framework helps to implement loose coupling between the classes.

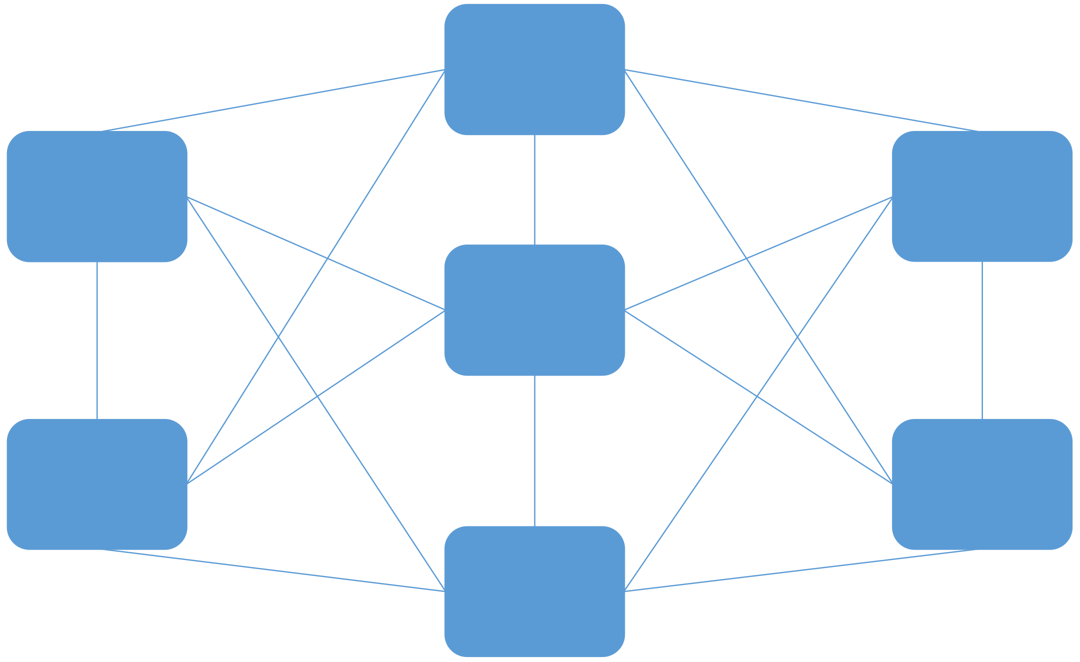
DEPENDENCY

Java application consist of many objects.



Objects using each other’s functionality or dependent on other object to perform its own functionality is called as dependency.

DEPENDENCY



Object dependencies will be **tightly coupled** in larger systems

Good design principle in object oriented programming concept suggest to break your application into reusable modules.

TIGHT COUPLING

```
class Circle{
 void drawShape()
              {   System.out.println(“Circle is drawn”);          }
}
class Triangle{
 void drawShape()
              {   System.out.println(“Triangle is drawn”);    }
}
Class Geometry{
   Triangle shape;
    void letsShape()
              {   shape= new Triangle();
                  shape.drawShape();          }
}
```

TIGHT COUPLING

```
Class Geometry{
    Circle shape;
    void letsShape(){
      shape= new Circle();
      shape.drawShape();
   }
 }
```

LOOSE COUPLING AND INTERFACE

```
interface Shape{
   void drawShape();
}
class Circle implements Shape{
  void drawShape()
              {     System.out.println(“Circle is drawn”);     }
}
class Triangle implements Shape {
  void drawShape()
              {     System.out.println(“Triangle is drawn”);  }
}
```

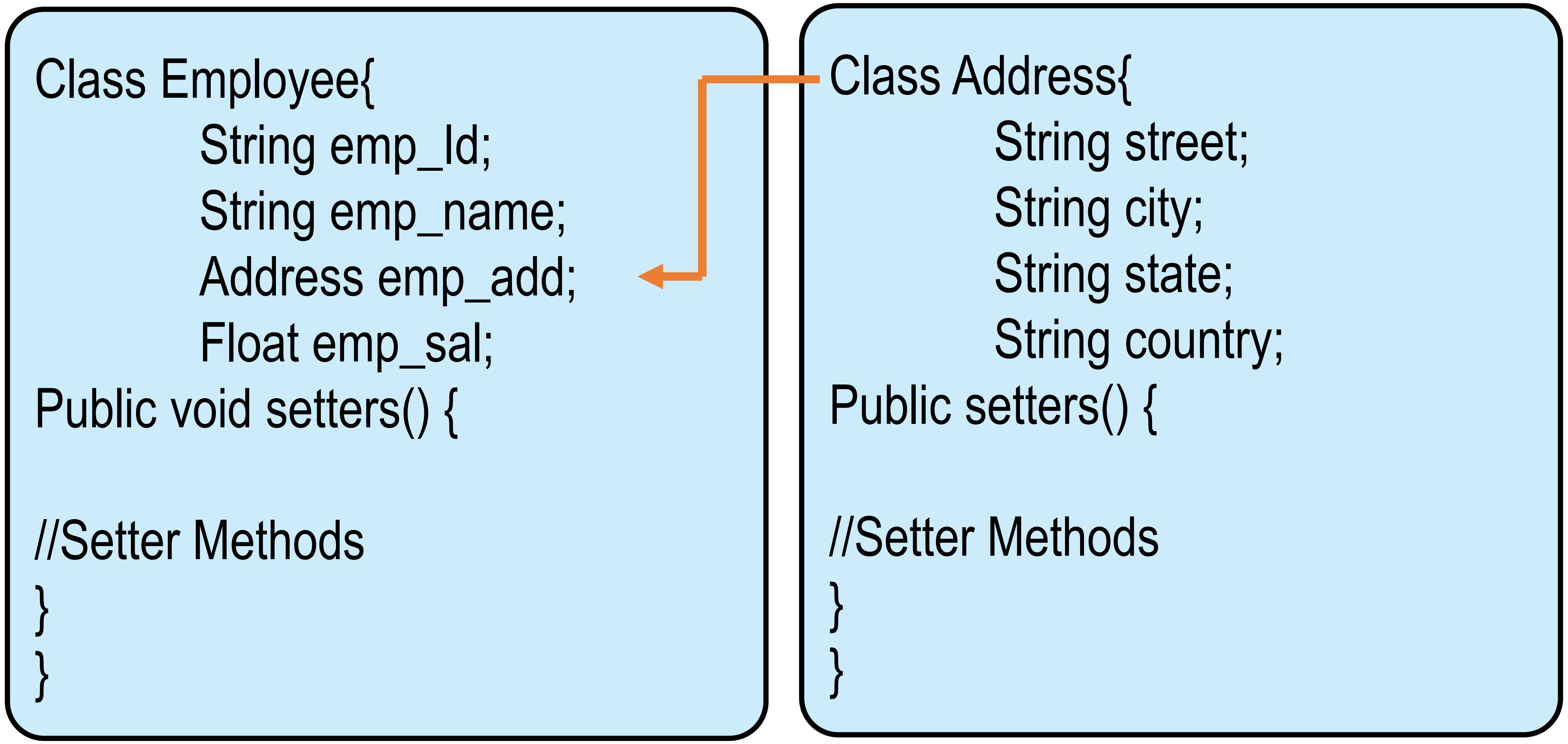
LOOSE COUPLING AND INTERFACE

```
Class Geometry{
  Shape shape;
   void letsShape(){
      shape= new Triangle();
      shape.drawShape();
   }
}
```

DEPENDENCY INJECTION - ANNOTATION

DEPENDENCY INJECTION

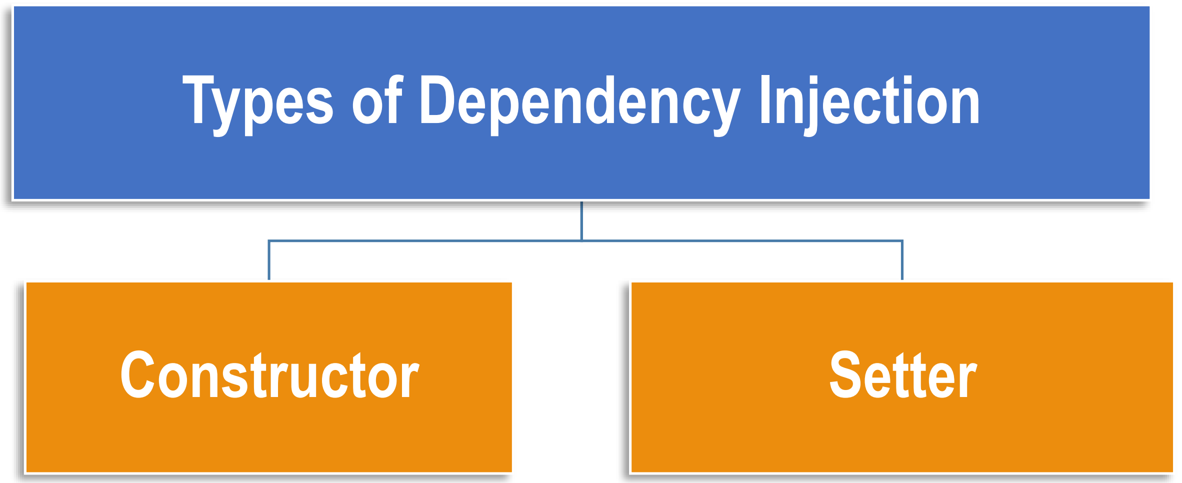
Example



```
Class Employee{
            String emp_Id;
            String emp_name;
            Address emp_add;
            Float emp_sal;
Public void setters() {

//Setter Methods
}
}
```

```
Class Address{
            String street;
            String city;
            String state;
            String country;
Public setters() {

//Setter Methods
}
}
```

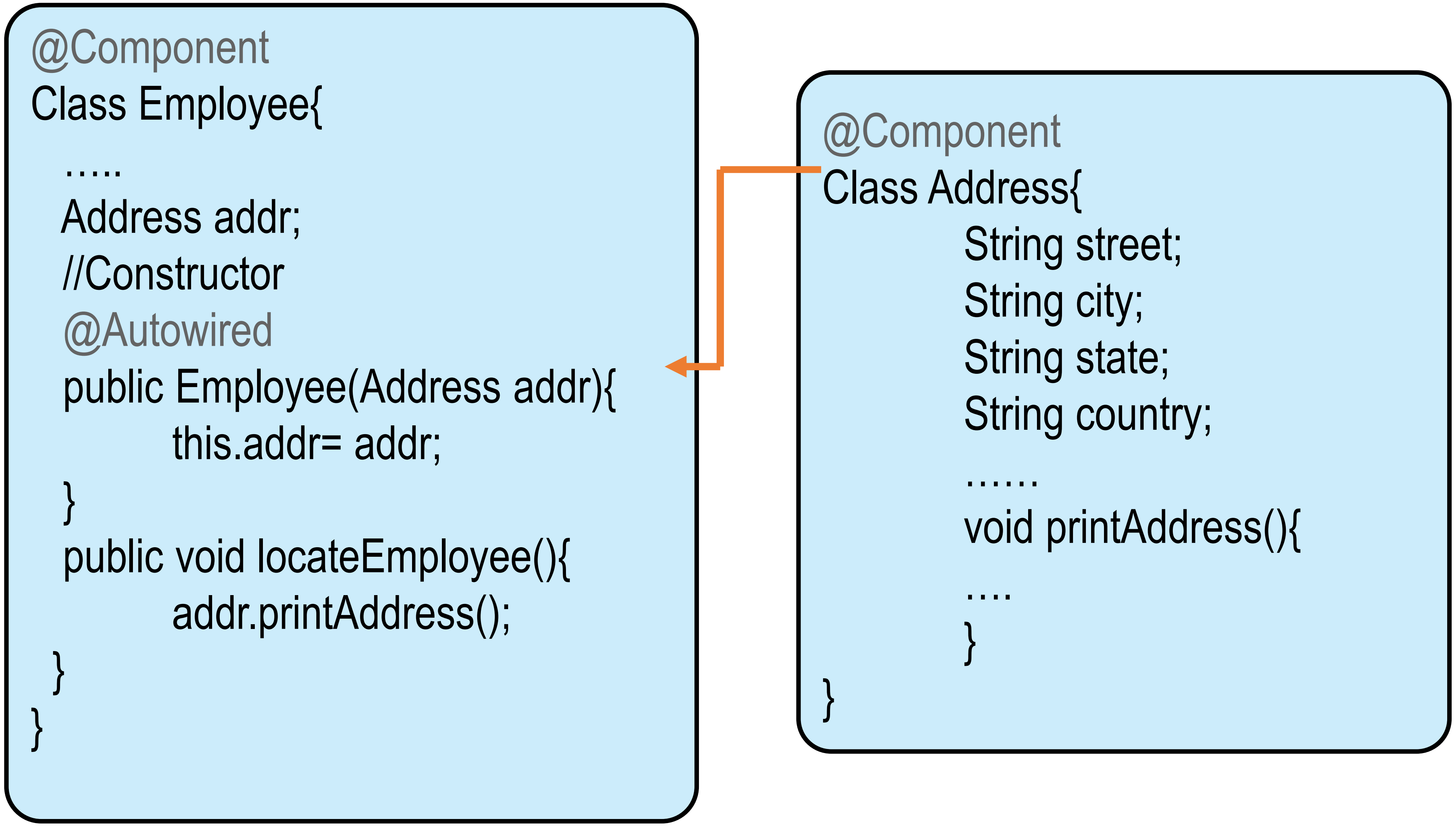
DEPENDENCY INJECTION



Autowiring → Inject the object dependency automatically

@Autowired → Used for automatic dependency injection

AUTOWIRING USING CONSTRUCTOR METHOD



```
@Component
Class Employee{
   …..
   Address addr;
   //Constructor
   @Autowired
   public Employee(Address addr){
            this.addr= addr;
   }
   public void locateEmployee(){
            addr.printAddress();
  }
}
```

```
@Component
Class Address{
            String street;
            String city;
            String state;
            String country;
            ……
            void printAddress(){
            ….
            }
}
```

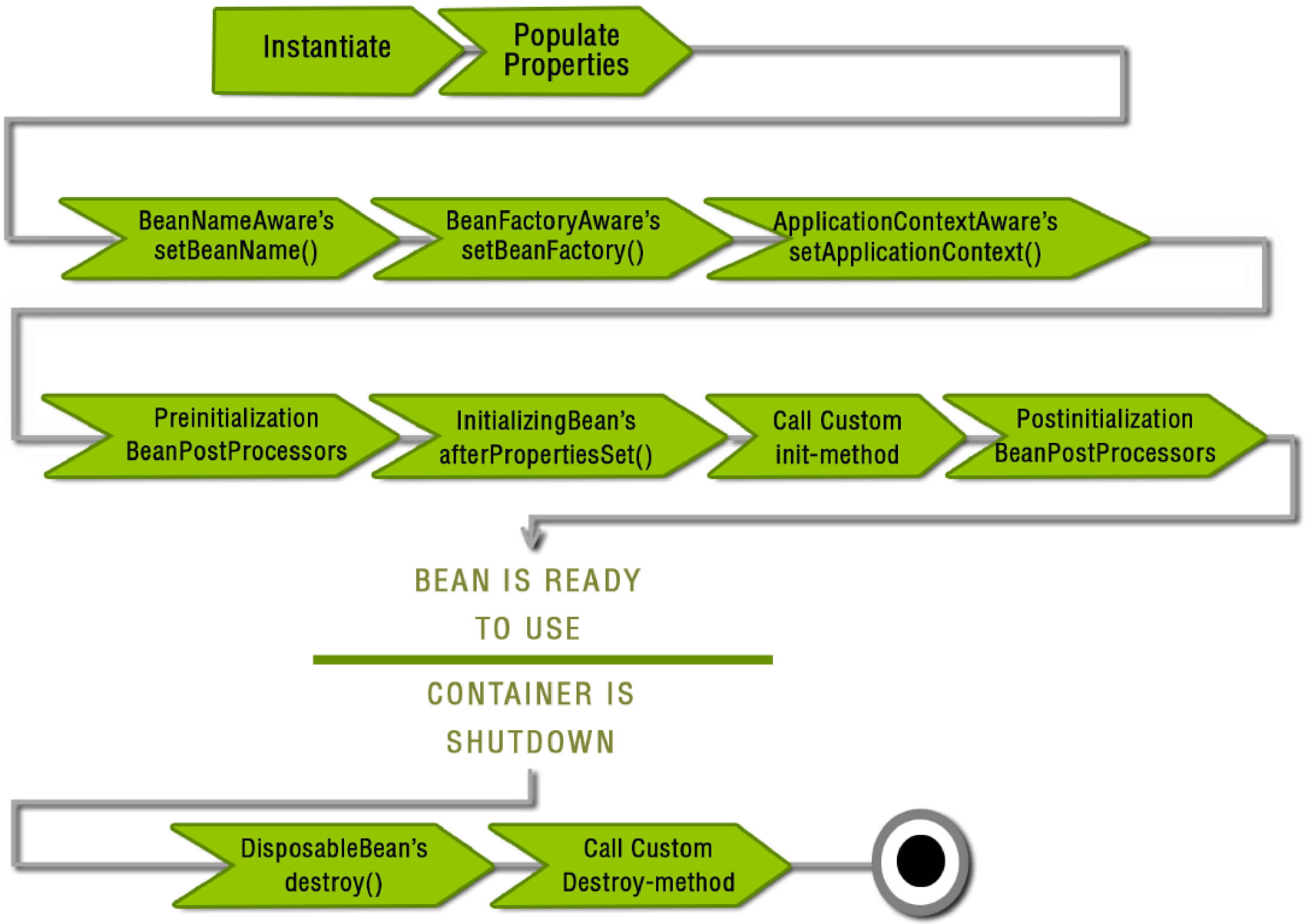
AUTOWIRING USING SETTER METHOD

```
@Component
Class Employee{
   .....
   Address addr;
   //Constructor
   @Autowired
   public setAddr(Address addr){
          this.addr= addr;
   }
   public void locateEmployee(){
          addr.printAddress();
  }
}
```

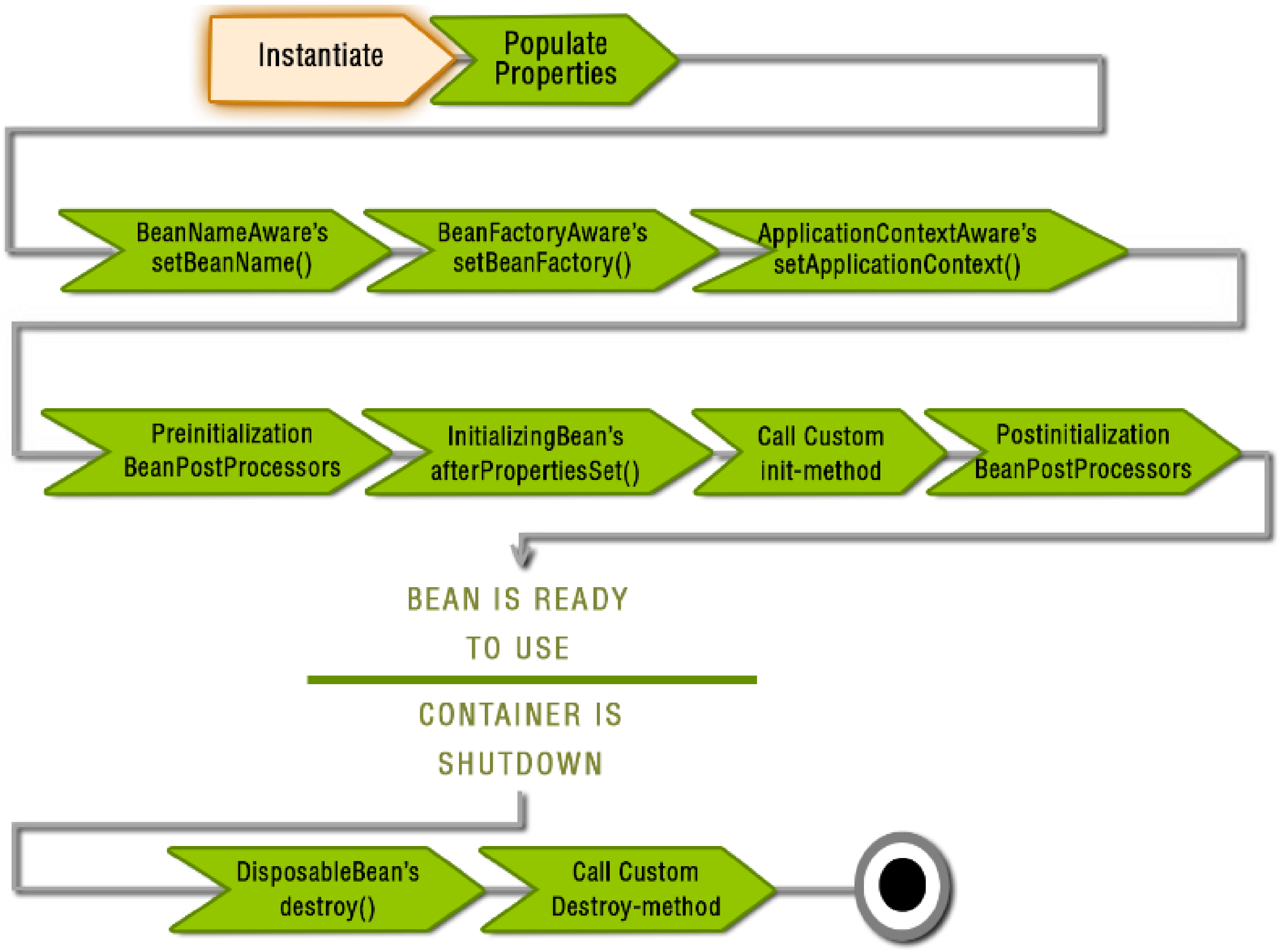
```
@Component
Class Address{
          String street;
          String city;
          String state;
          String country;
          ......
          void printAddress(){
          ....
          }
}
```

SPRING BEAN LIFE CYCLE

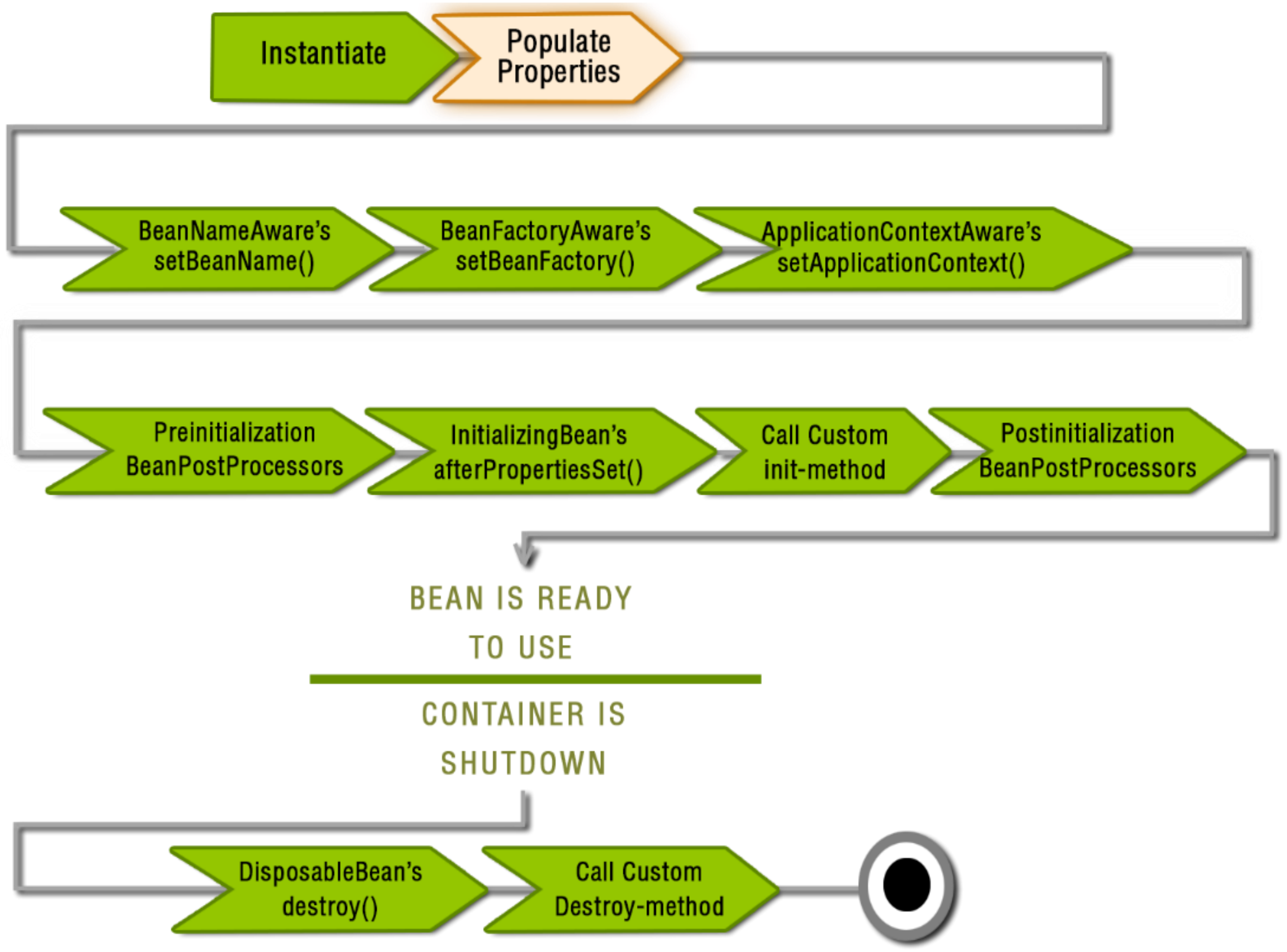
LIFE CYCLE OF SPRING BEAN


Instantiate
Populate Properties
BeanNameAware's setBeanName()
BeanFactoryAware's setBeanFactory()
ApplicationContextAware's setApplicationContext()
Preinitialization BeanPostProcessors
InitializingBean's afterPropertiesSet()
Call Custom init-method
Postinitialization BeanPostProcessors
BEAN IS READY TO USE
CONTAINER IS SHUTDOWN
DisposableBean's destroy()
Call Custom Destroy-method

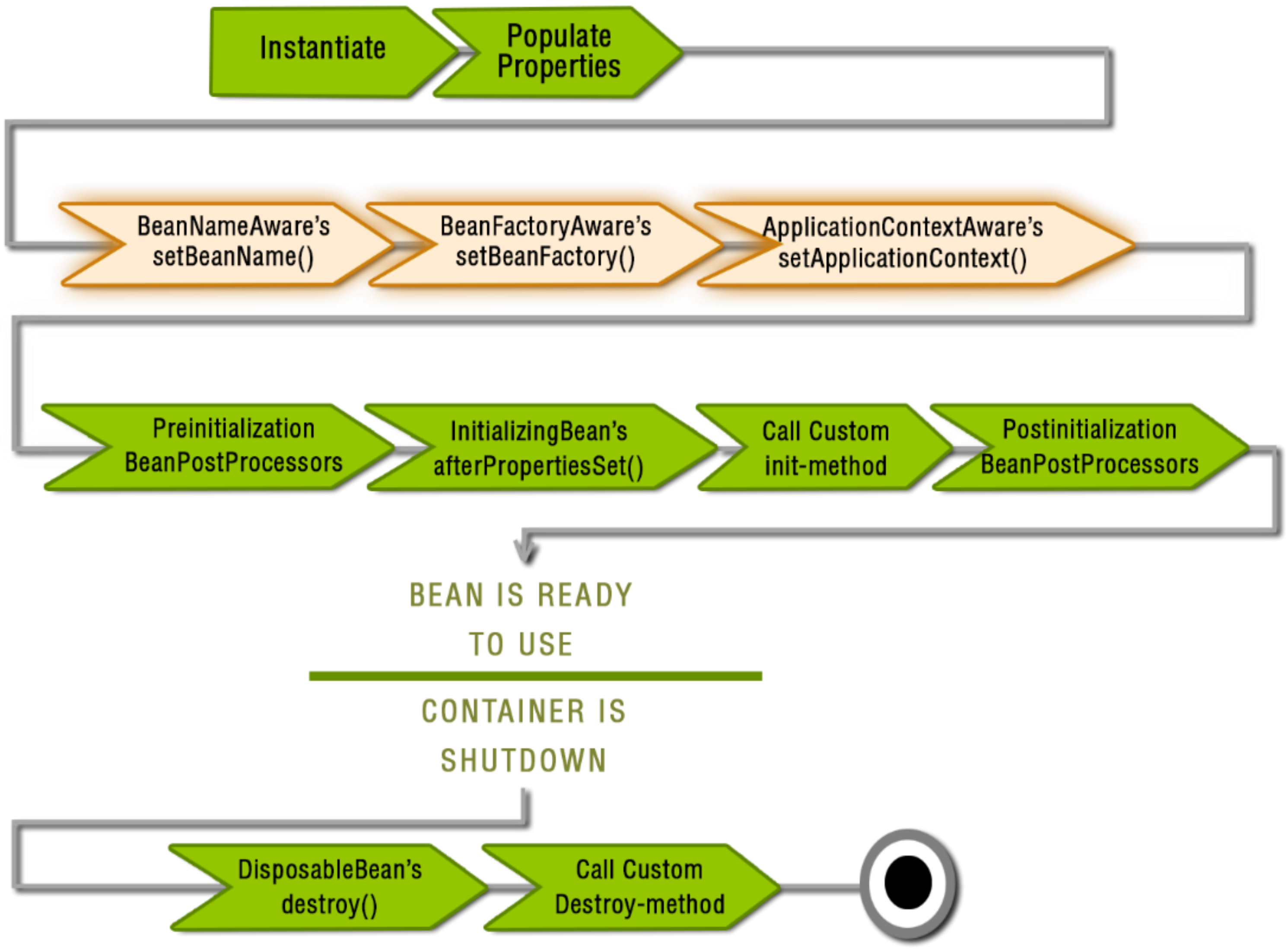
LIFE CYCLE OF SPRING BEAN


Instantiate
Populate Properties
BeanNameAware's setBeanName()
BeanFactoryAware's setBeanFactory()
ApplicationContextAware's setApplicationContext()
Preinitialization BeanPostProcessors
InitializingBean's afterPropertiesSet()
Call Custom init-method
Postinitialization BeanPostProcessors
BEAN IS READY TO USE
CONTAINER IS SHUTDOWN
DisposableBean's destroy()
Call Custom Destroy-method

LIFE CYCLE OF SPRING BEAN

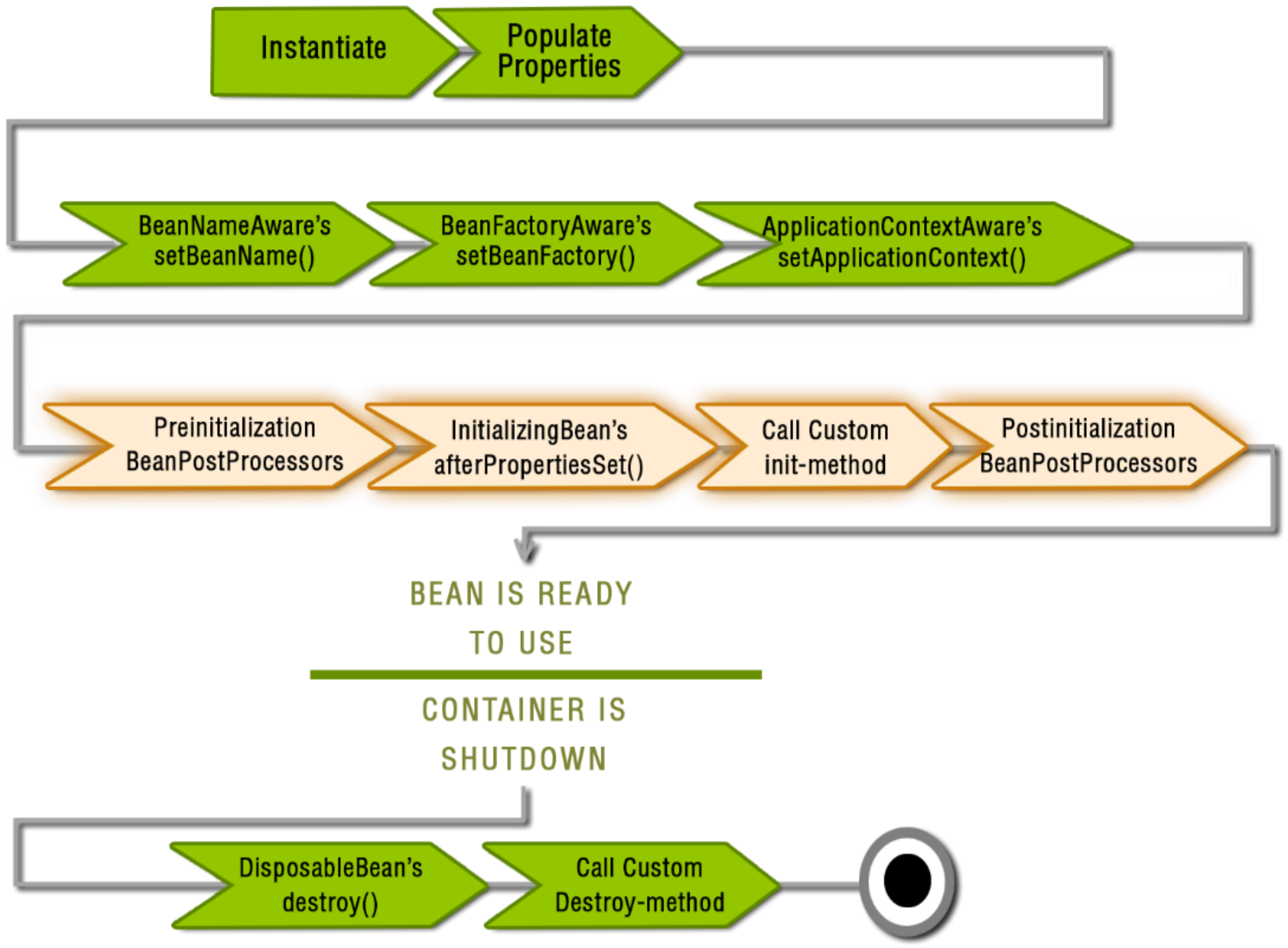

Instantiate
Populate Properties
BeanNameAware's setBeanName()
BeanFactoryAware's setBeanFactory()
ApplicationContextAware's setApplicationContext()
Preinitialization BeanPostProcessors
InitializingBean's afterPropertiesSet()
Call Custom init-method
Postinitialization BeanPostProcessors
BEAN IS READY TO USE
CONTAINER IS SHUTDOWN
DisposableBean's destroy()
Call Custom Destroy-method

LIFE CYCLE OF SPRING BEAN

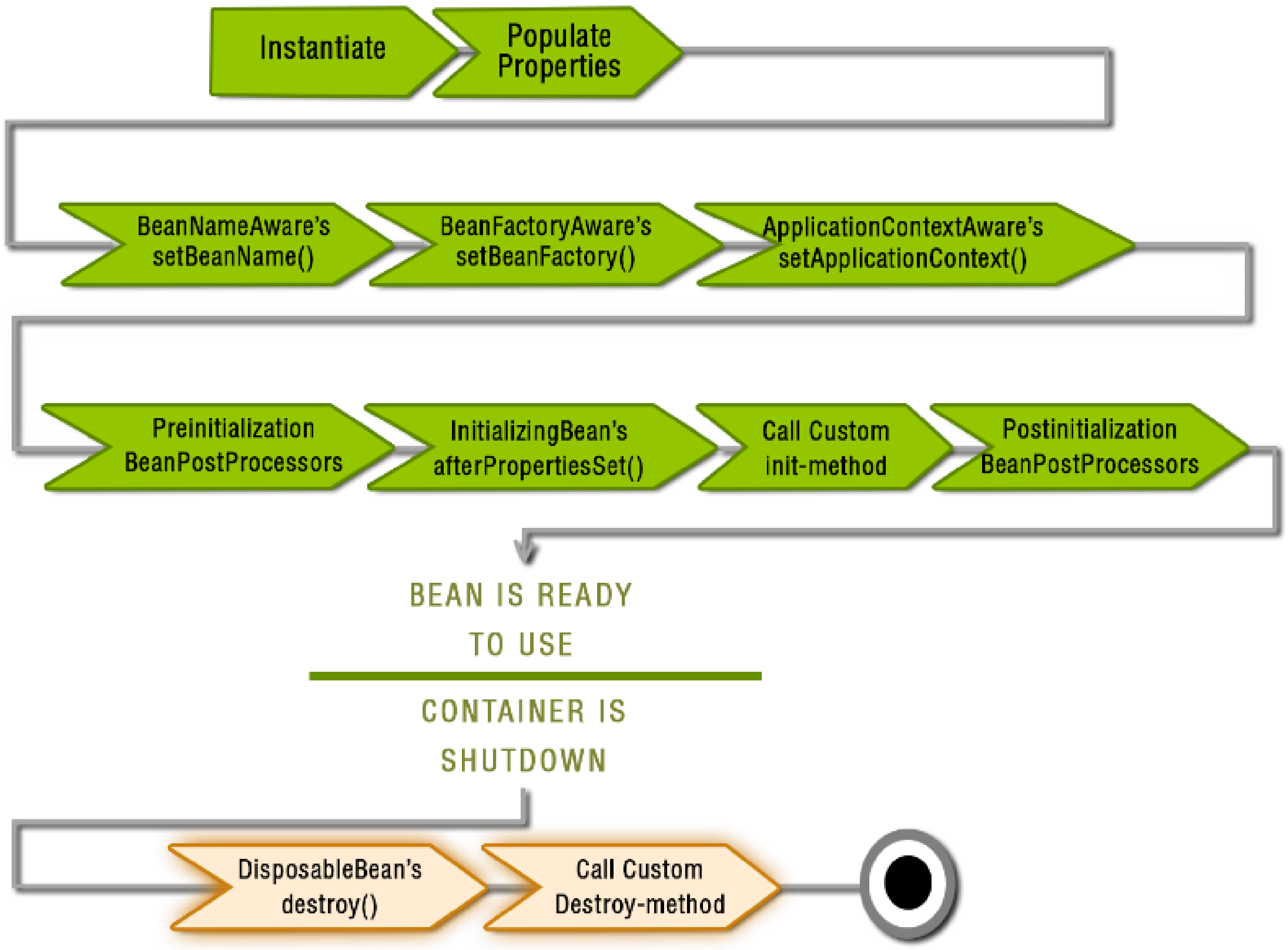


Instantiate
Populate Properties
BeanNameAware's setBeanName()
BeanFactoryAware's setBeanFactory()
ApplicationContextAware's setApplicationContext()
Preinitialization BeanPostProcessors
InitializingBean's afterPropertiesSet()
Call Custom init-method
Postinitialization BeanPostProcessors
BEAN IS READY TO USE
CONTAINER IS SHUTDOWN
DisposableBean's destroy()
Call Custom Destroy-method

LIFE CYCLE OF SPRING BEAN

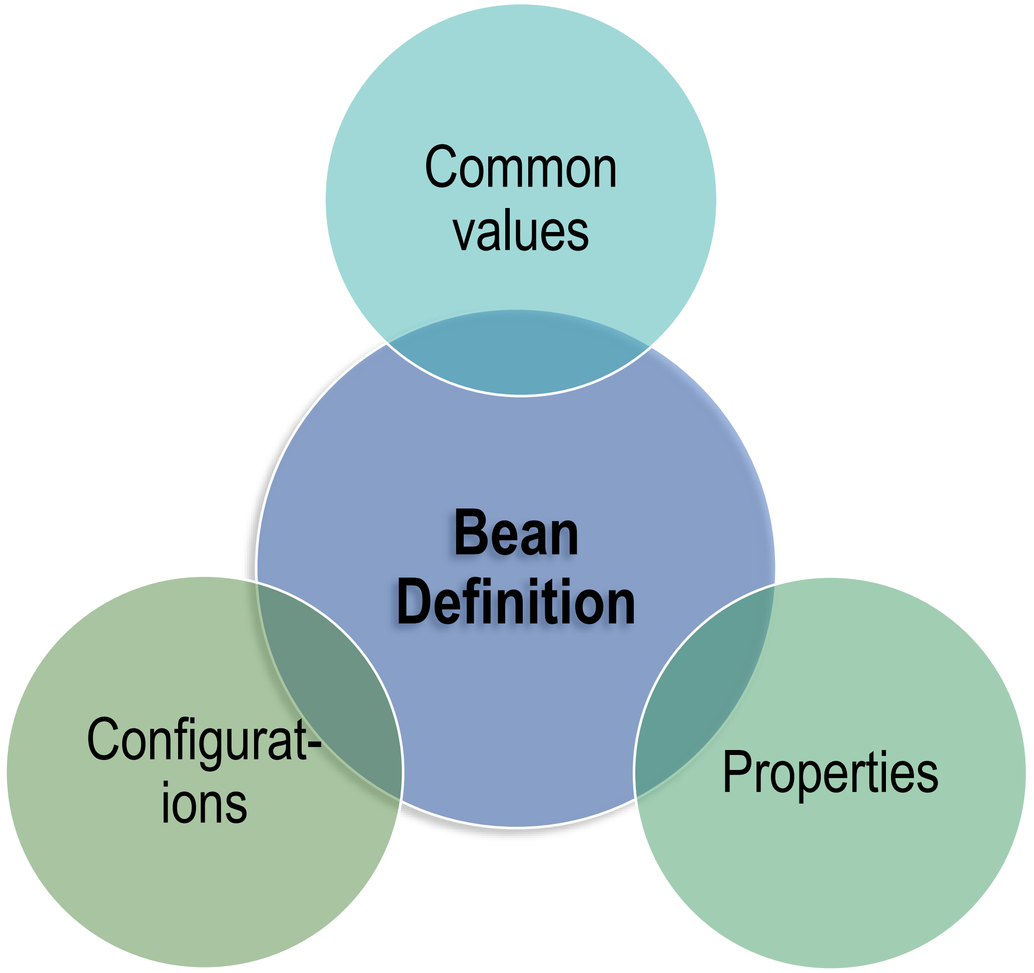

Instantiate
Populate Properties
BeanNameAware's setBeanName()
BeanFactoryAware's setBeanFactory()
ApplicationContextAware's setApplicationContext()
Preinitialization BeanPostProcessors
InitializingBean's afterPropertiesSet()
Call Custom init-method
Postinitialization BeanPostProcessors
BEAN IS READY TO USE
CONTAINER IS SHUTDOWN
DisposableBean's destroy()
Call Custom Destroy-method

LIFE CYCLE OF SPRING BEAN

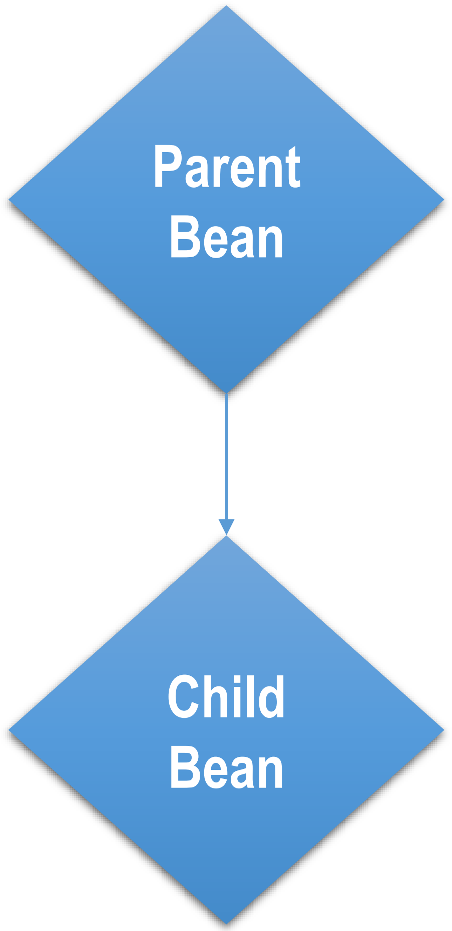

Instantiate
Populate Properties
BeanNameAware's setBeanName()
BeanFactoryAware's setBeanFactory()
ApplicationContextAware's setApplicationContext()
Preinitialization BeanPostProcessors
InitializingBean's afterPropertiesSet()
Call Custom init-method
Postinitialization BeanPostProcessors
BEAN IS READY TO USE
CONTAINER IS SHUTDOWN
DisposableBean's destroy()
Call Custom Destroy-method

BEAN DEFINITION INHERITANCE AND INNER BEAN

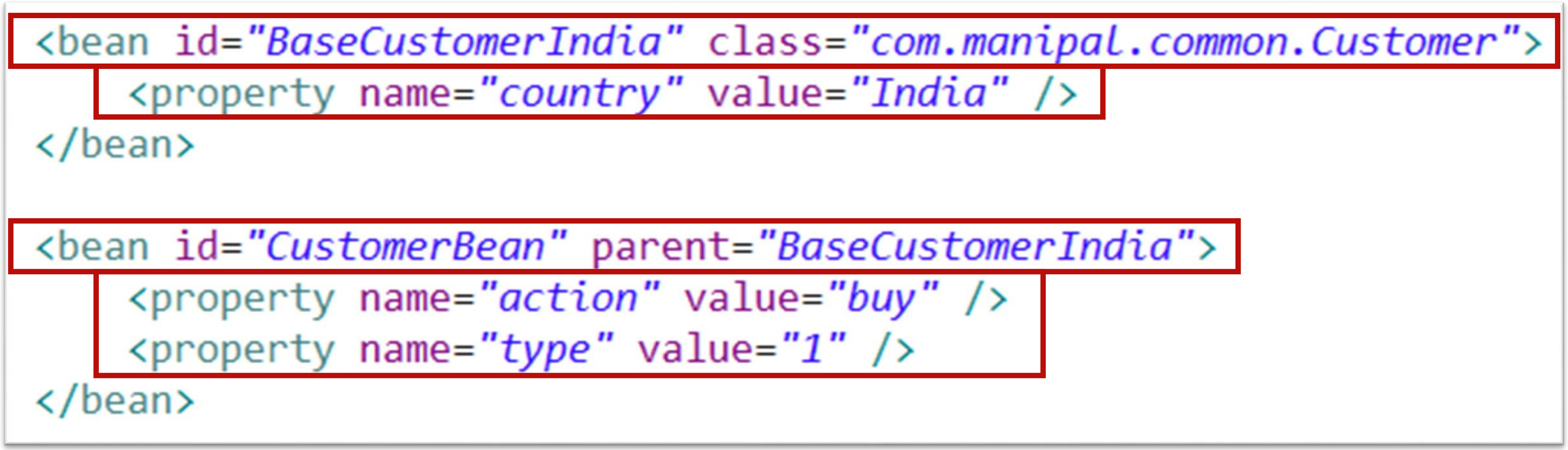
BEAN DEFINITION INHERITANCE



BEAN DEFINITION INHERITANCE



BEAN DEFINITION INHERITANCE: EXAMPLE



```
<bean id="BaseCustomerIndia" class="com.manipal.common.Customer">
    <property name="country" value="India" />
</bean>

<bean id="CustomerBean" parent="BaseCustomerIndia">
    <property name="action" value="buy" />
    <property name="type" value="1" />
</bean>
```

INNER BEAN

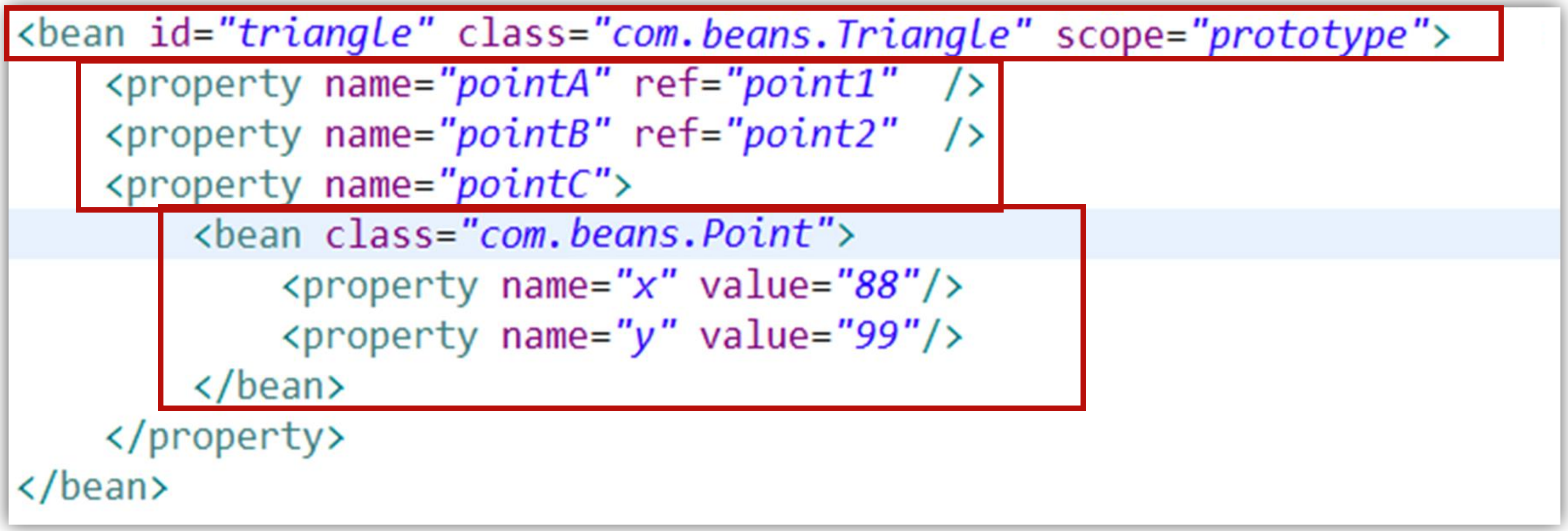
Beans which are defined within the scope of another bean

Supported by setter and constructor injection

Declared as a <bean> element inside a <property> or <constructor-arg> element

ID or name attributes are optional

INNER BEAN: EXAMPLE



```
<bean id="triangle" class="com.beans.Triangle" scope="prototype">
    <property name="pointA" ref="point1"  />
    <property name="pointB" ref="point2"  />
    <property name="pointC">
        <bean class="com.beans.Point">
            <property name="x" value="88"/>
            <property name="y" value="99"/>
        </bean>
    </property>
</bean>
```