Universidad ICESI

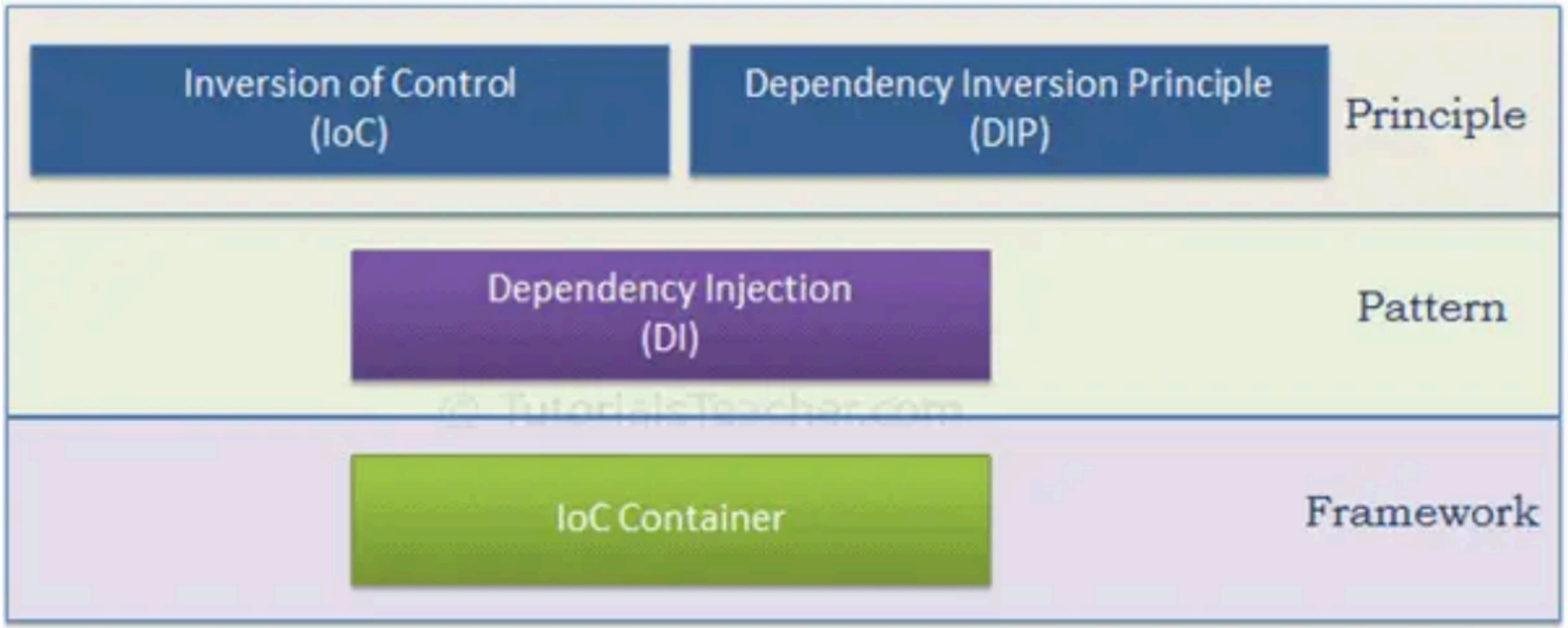
45 años

Spring Introduction

Computación en Internet II
2025-1

Frameworks often implement *design patterns and principles* to help you write better code.

- **Design Principle:** Provide high level guidelines to design better software applications. They do not provide implementation guidelines and are not bound to any programming language. The SOLID (SRP, OCP, LSP, ISP, DIP) principles are one of the most popular sets of design principles.
- **Design Pattern:** Provides low-level solutions related to implementation, of commonly occurring object-oriented problems. In other words, design pattern suggests a specific implementation for the specific object-oriented programming problem. Ex: Singleton, factory, Command, Observer, etc.


Inversion of Control
(IoC)
Dependency Inversion Principle
(DIP)
Principle
Dependency Injection
(DI)
Pattern
IoC Container
Framework

Frameworks often implement design patterns and principles such as:

- Model-View-Controller (MVC) Framework, which separates the application logic into three components with single responsibilities.

- Dependency Injection (DI) Framework allows you to inject dependencies into your classes or modules instead of creating them inside, reducing coupling and increasing cohesion.

- Test-Driven Development (TDD) Framework encourages writing tests before code to help follow the SRP and refactor the code for greater cohesion and less coupling.

What is Spring?

- The **spring framework** provides a programming and configuration mode for java-based enterprise applications.

- Spring focuses on the “plumbing” of enterprise applications so that teams can focus on application-level business logic.

What is Spring?

- *A container:*
  - Create objects and makes them available to your application.
  - Uses a declarative XML structure to define components that “live” in the container.

- *A Framework:*
  - Provides an infrastructure of classes that make it easier to accomplish tasks.
  - For example, Templates provides an abstraction away from details of dataAccess, remote calling.

What is Spring?

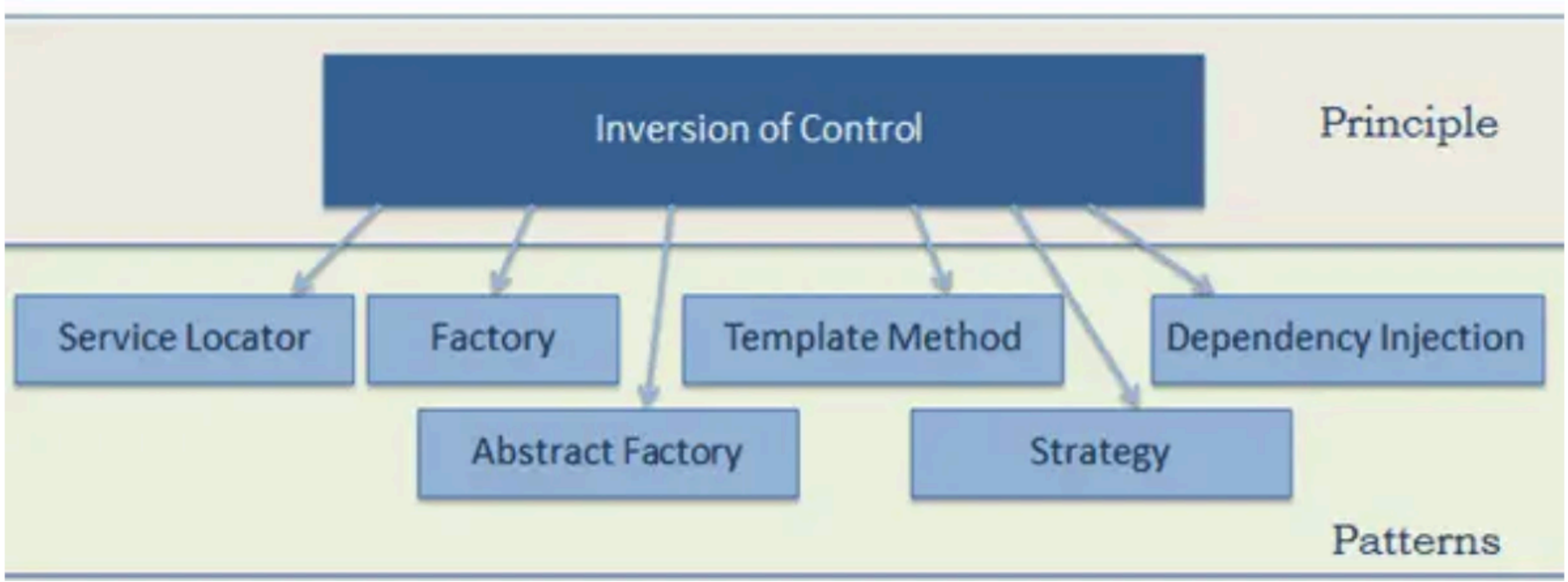
- Spring is a lightweight framework that addresses each ***tier*** in a Web application
  - **Presentation Layer:** An *MVC framework* that is most similar to *Struts* but is more powerful and easier to use.
  - **Business layer**: Lightweight *IoC container* and *AOP* support
  - **Persistence layer**: *DAO template* support for popular *ORMs* and *JDBC*

Inversion of Control

- The basic concept of the inversion of Control principle is that programmers don’t need to create their objects but instead, they need to describe how they should be created through rules defined in e.g. factory classes and/or , configuration files.

- Subsequently, your application code does not directly connect your implementation components together but instead uses abstractions such as interfaces that represent them. Spring provides the actual implementations.

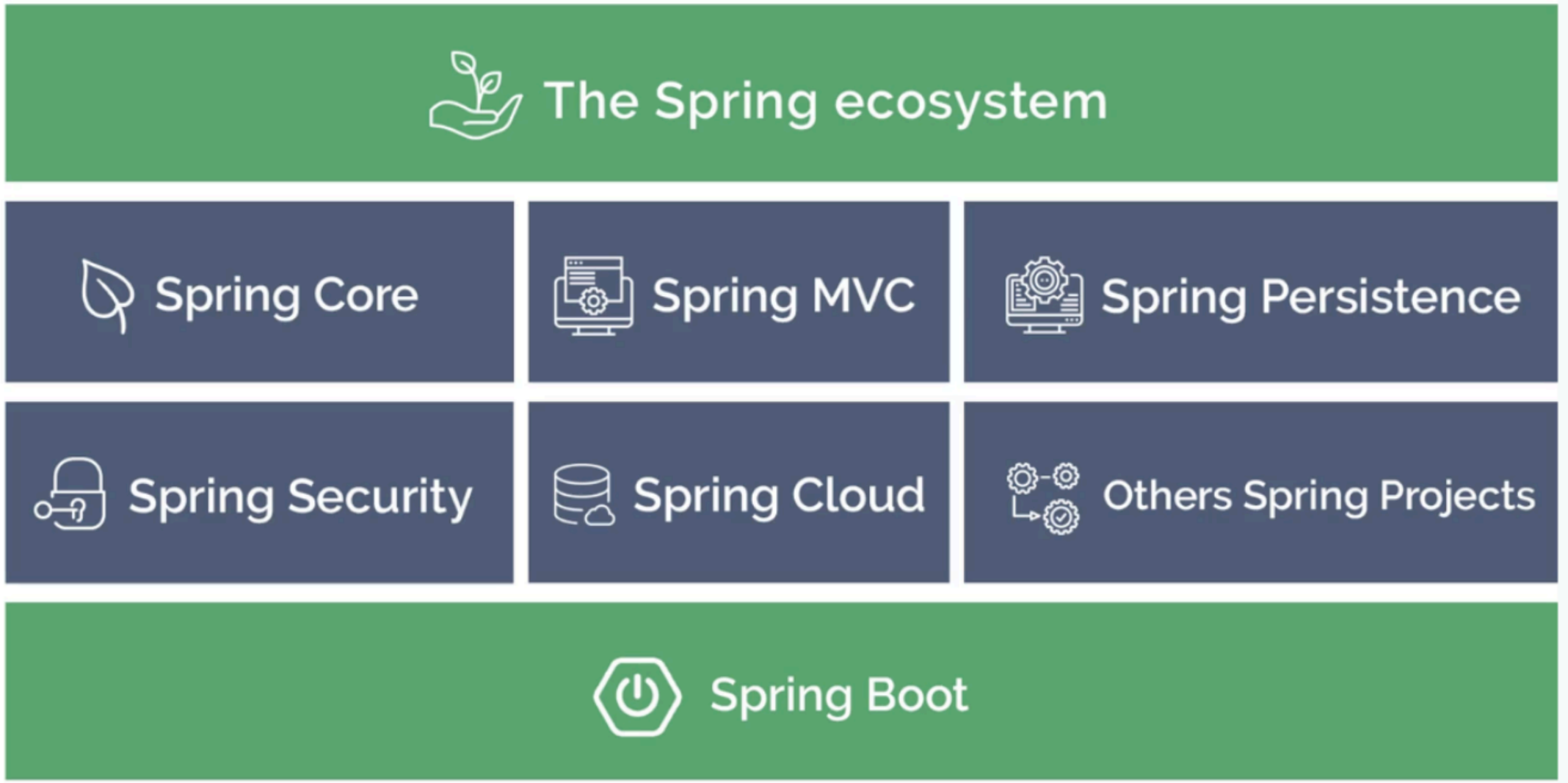
The following pattern (but not limited) implements the IoC principle.



Dependency Injection

- Dependency injection is a pattern used to create instances of objects that have delegation patterns to other objects. However, at compile time the “calling class” has no knowledge of the dependency class implementation it will delegate to at Runtime. Because our Delegator Class code is coded to the interface abstraction of the delegating class.

- With Spring, the implementation of dependency injection can be achieved via setter and/or constructor injection techniques.


The Spring ecosystem
Spring Core
Spring MVC
Spring Persistence
Spring Security
Spring Cloud
Others Spring Projects
Spring Boot

Core modules

- The Spring framework comprises of many modules such as cores, beans, context, expression languages, AOP, Aspects, Instrumentation, JDBC, ORM, OXM, JMS, Transaction, Web, Servlet, Struts etc.

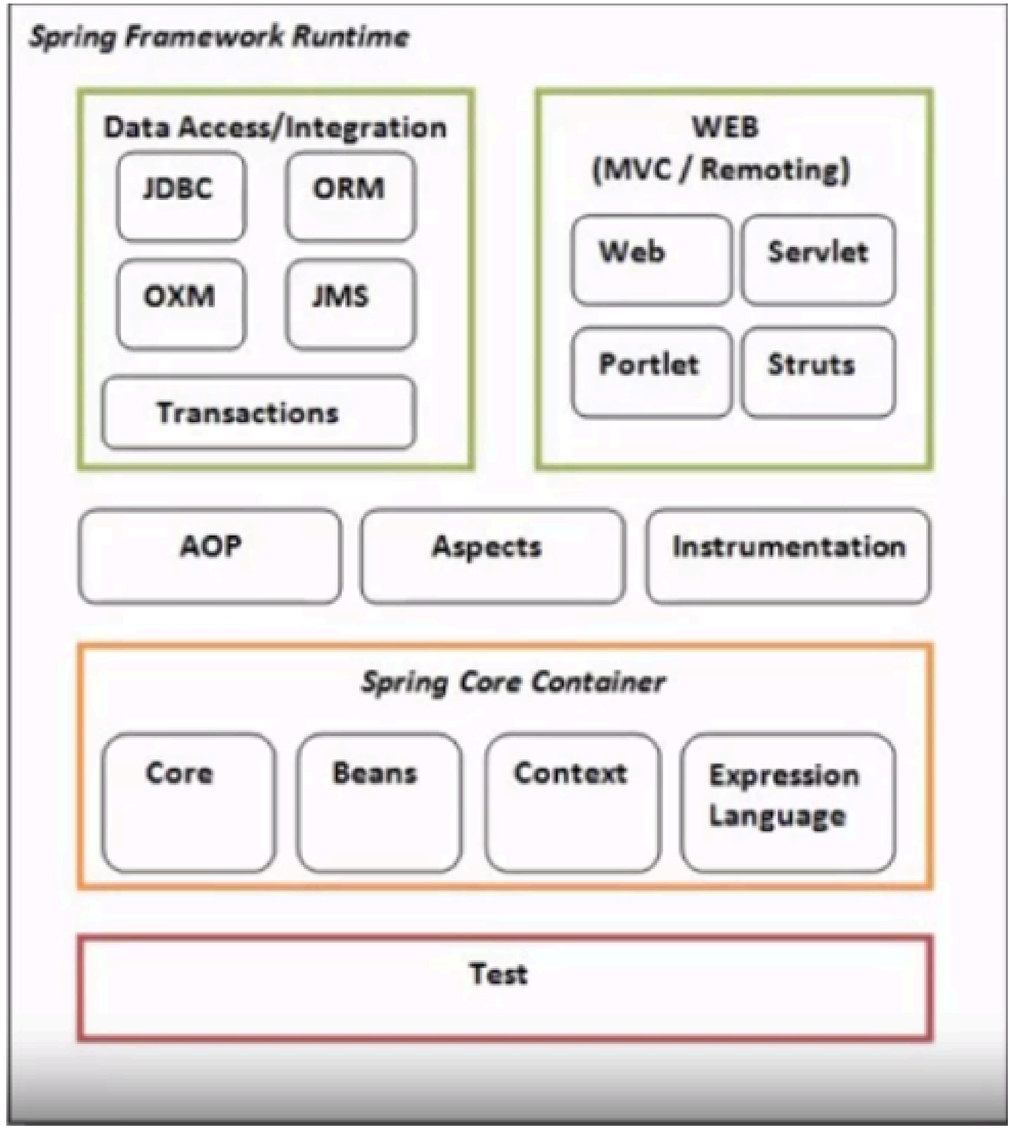
The *Spring core container* contains core, beans, context and expression language (EL) Modules.

Core modules

These modules are grouped into:

- Test
- Core Container
- Aspect Oriented Programming

- InstrAspectsumentation
- Data Access / Integration
- Web (MVC / Remoting)

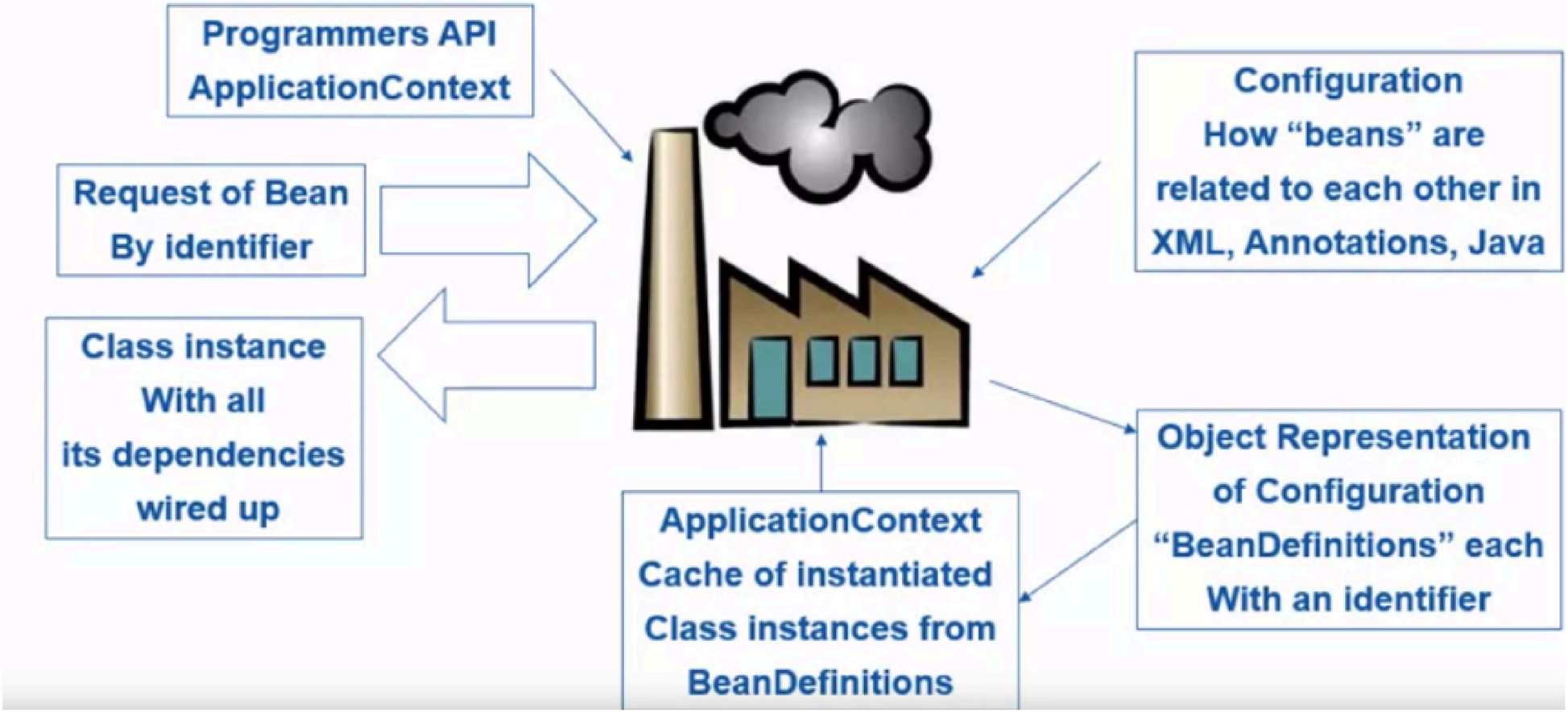


Core modules

- **Core and Beans:** these modules provide IOC and Dependency injection features
- **Context:** This module supports internationalization (I18N), EJB, JMS.
- **Expression Language:** It is an extension to the EL defined in JSP. It provides support to setting and getting property values and even method invocation
- **AOP, Aspects and Instrumentation:** These modules support aspect oriented programming implementation where you can use Advices, Pointcuts
- The aspects module provides support to integration with AspectJ
- **Data Access / Integration:** These modules basically provide support to interact with the database
- **Web:** This group provide support to create web applications

Spring is like a Factory

It creates your class instances through Configuration



Spring Wires up Beans (Class instances)

The plumbing code is done for you

No Spring, you assemble the product yourself

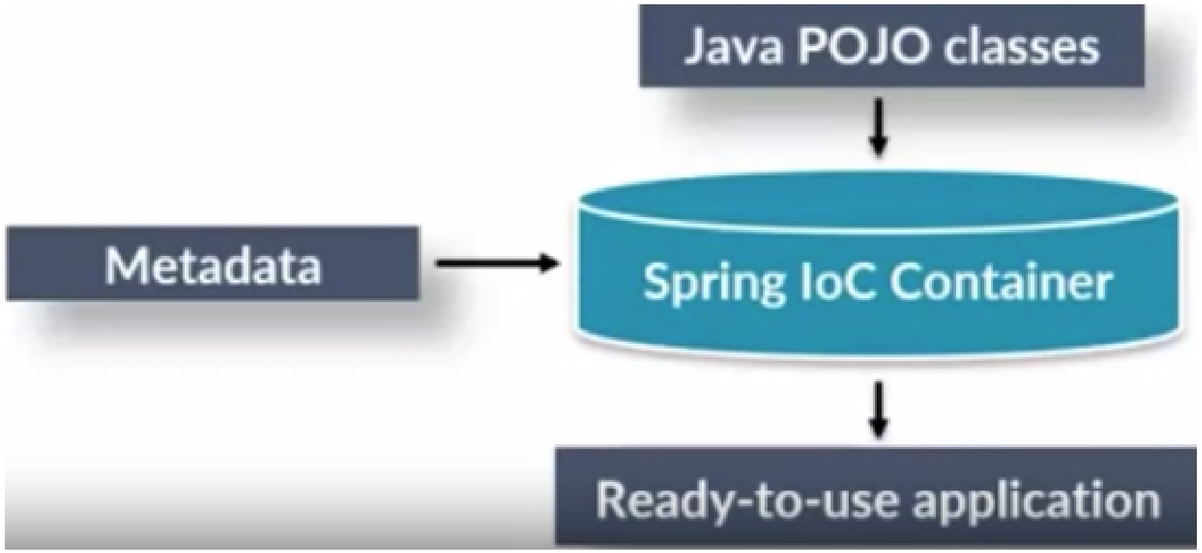
Using Spring, it assembles the product for you, via configuration, and is ready for use

Discussion: Spring and Plumbing

- Explain what is meant by "Spring wires up the plumbing code".
- Why do you think this is an important aspect of Spring?
- Would this appeal to many developers?

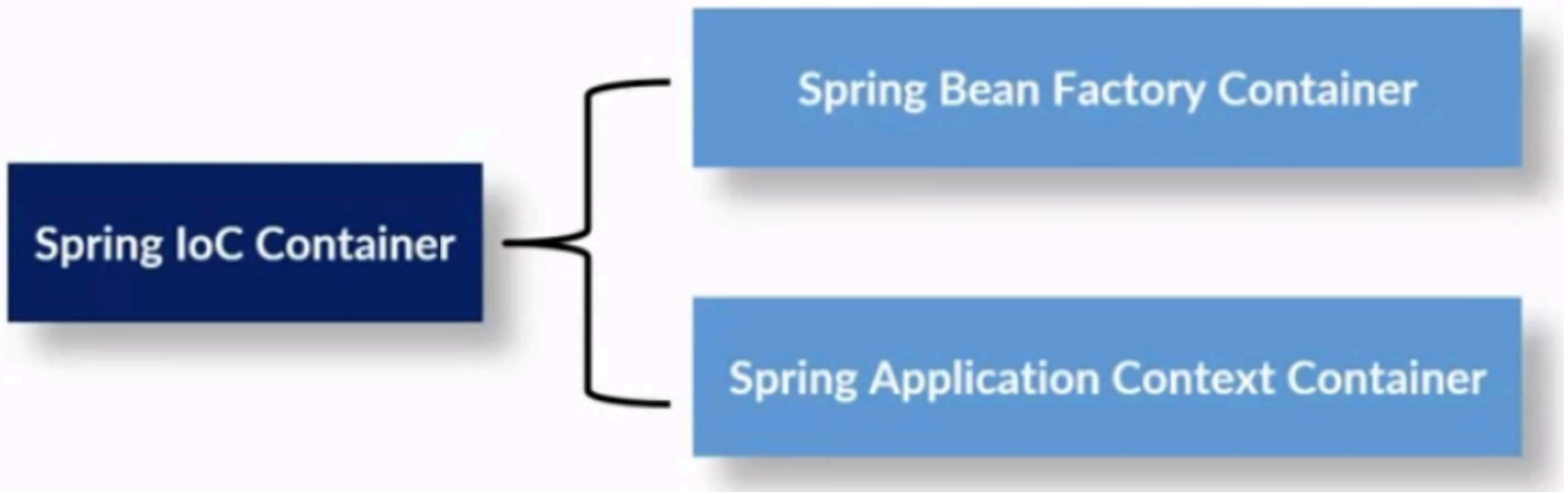
Spring IoC Containers

- The Spring Containers forms the core of the Spring Framework
- The container instantiates, configures, and assembles the dependencies between the objects.
- The IoC container construct an object of the selected class.
- And also inject the dependency objects via a constructor, a property or a function at execution time and disposes it at suitable time.



The Spring Container

- Spring containers are simple java classes and interfaces.
- There are many spring containers among which two are very important.



- **Bean factories defined** by the org.springframework.beans.factory.BeanFactory interface are the simplest of containers, providing basic support for dependency injection (DI). This BeanFactory is the super interface of all the spring containers.
- **Application contexts** defined by the org.springframework.context.ApplicationContext interface build on the notion of a bean factory by providing application framework services.

The **ApplicationContext** container includes all functionality of the BeanFactoryContainer, so it is generally recommended over BeanFactory.

The Spring Container

- These **BeanFactory** and **ApplicationContext** are java interfaces, but in spring world these are called as containers.

- *BeanFactory* is an implementation of the Factory design pattern. As the name says it is a factory of Beans whose responsibility is to create and dispense beans.

- *ApplicationContext* interface extends the BeanFactory interface. To take full advantage of Spring framework most of the time we load beans using more advanced spring container ApplicationContext.

Spring Bean

- Spring beans are the objects which are created and managed completely by spring container.
- These beans are the heart of the application
- Beans can be defined in spring either by using XML configuration or by using Annotation.
- In XML configuration, bean can be defined using <bean> tag inside <beans> tag.
- In Annotation configuration, bean can be defined using the annotations,
- Like @Component, @Service, @Controller and @Repository on top of the class definition.
- We can also achieve beans definition completely in java using java configuration.

Spring Bean Scope

- Scope of a bean specifies what kind of object has to be created by container for bean defined.
- We can define the scope of the bean while defining the bean in the spring configuration file
- We can also customize and create custom scope.
- Types of bean scopes supported by Spring are as follows:
  - Singleton
  - Prototype
  - Request
  - Session
  - Application

Singleton Scope

- Singleton bean will be only one instance for entire spring container, no matter how many times and in how many places we access it using context.getBean() method.
- This single instance is stored in a cache of such singleton beans, and all subsequent requests and references for that named bean return the cached object.
- If we didn’t add any scope for the bean, then Spring automatically makes that bean as Singleton.

```
< !--A bean definition with singleton scope-- >

<bean id="..." class="..." scope="singleton">
    <!-- collaborators and configuration for this bean go here -->
</bean>
```

Prototype Scope

- Prototype scope means container creates new instance every time we ask container to provide the bean using getBean() method.
- Prototype scope must be defined explicitly.
- In simple words, each context.getBean() method returns a new object.

```
< !--A bean definition with prototype scope-- >

<bean id="..." class="..." scope="prototype">
    <!-- collaborators and configuration for this bean go here -->
</bean>
```

Request Scope

- In Request Scope, container creates a new instance for each and every HTTP request.
- So, If server is currently handling 50 requests, then container can have at most 50 individual instances of bean class.
- Any state change to one instance, will not be visible to other instances.
- These instances are destructed as soon as request is completed.

```
@Component
@Scope("request")
public class BeanClass {
}
//or
@Component
@RequestScope
public class BeanClass {
}
```

Session Scope

- **In Session Scope, container creates a new instance for each and every HTTP request.**
- So, if server has 20 active sessions,
- then container can have at most 20 individual instances of bean class.
- All HTTP requests within single session will have access to same single bean instance.
- Any state change to one instance, will not be visible to other instances.

```
@Component
@Scope("session")
public class BeanClass {
}
//or
@Component
@SessionScope
public class BeanClass {
}
```

Application Scope

- **In Application Scope, container creates one instance per web application runtime.**
- It is almost similar to singleton scope, with only two differences –
- Application scoped bean is singleton per ServletContext, whereas singleton scoped bean is singleton per ApplicationContext.
- Application scoped bean is visible as a ServletContext attribute.

```
@Component
@Scope("application")
public class BeanClass {
}
//or
@Component
@ApplicationScope
public class BeanClass {
}
```

WebSocket Scope

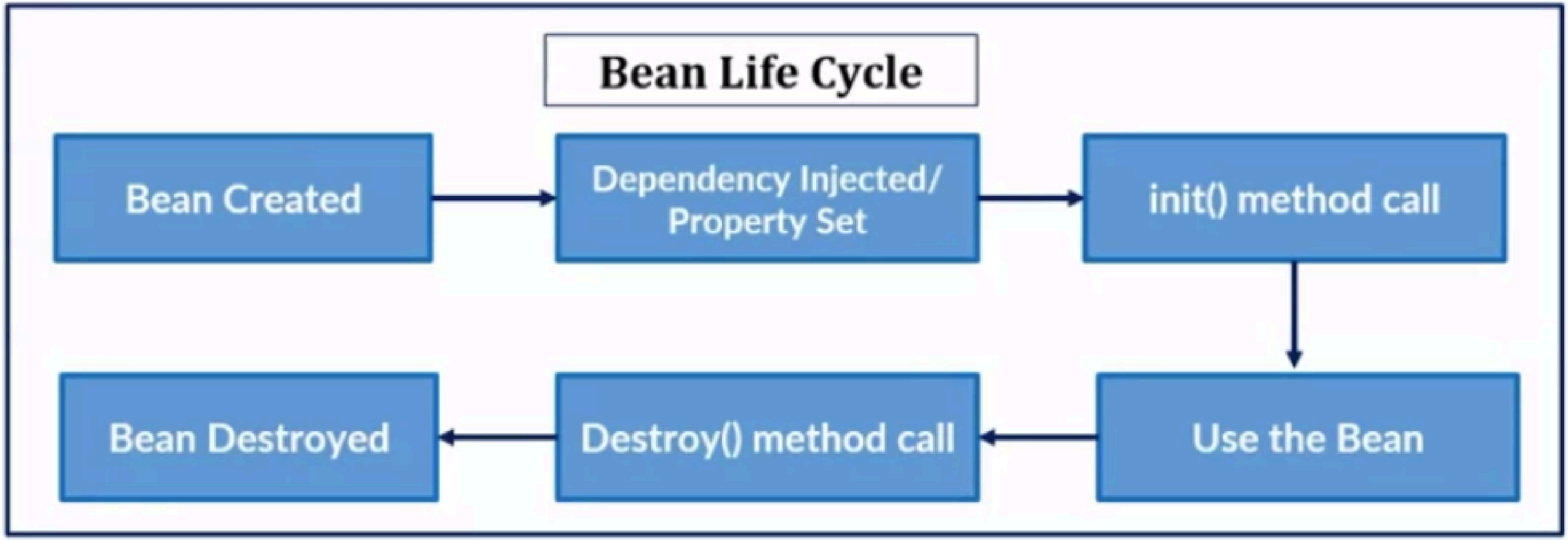
- The WebSocket Protocol enables two-way communication between a client and a remote host that has opted-in to communication with client.
- WebSocket Protocol provides a single TCP connection for traffic in both directions.
- This is useful for multi-user applications with simultaneous editing and multi-user games.

```
@Component
@Scope("websocket")
public class BeanClass {
}
```

Spring Bean Lifecycle

- IOC container is responsible for creating objects from configuration metadata that we supply in a config file.
- The container is responsible to manage the overall life cycle of the Spring beans,
- i.e from the creation of a bean to its destruction.
- **There are 3 ways in which we can implement Bean Life Cycle:**
  - By using XML Based Configuration
  - By using Annotation Based Configuration
  - By using Java Based Configuration.

Spring Bean Lifecycle



Spring Bean Lifecycle

HOW TO CUSTOMIZE THE BEAN LIFECYCLE ?

Spring framework provides the following four ways for controlling life cycle events of a bean:

- InitializingBean and DisposableBean callback interfaces.
- Aware interfaces for specific behavior.
- Custom init() and destroy() methods in bean configuration file.
- @PostConstruct and @PreDestroy annotations.