UNIVERSITÀ DELLA CALABRIA

DIPARTIMENTO DI **MATEMATICA E INFORMATICA**

AGILE SOFTWARE DEVELOPMENT

Dr. Francesco Scalzo - francesco.scalzo@unical.it
Enterprise Solution Engineer at **Revelis s.r.l.**

Student Reception & info: send a mail

Spring Project

PRESENTATION
- Angular

REST
- Spring - @Controller

BUSINESS
- Spring - @Service

DATA ACCESS
- Spring – JPA, @Repository

DATA
- DB – MariaDB / HSQLDB

- https://www.jetbrains.com/idea/download/
- https://start.spring.io/

Spring

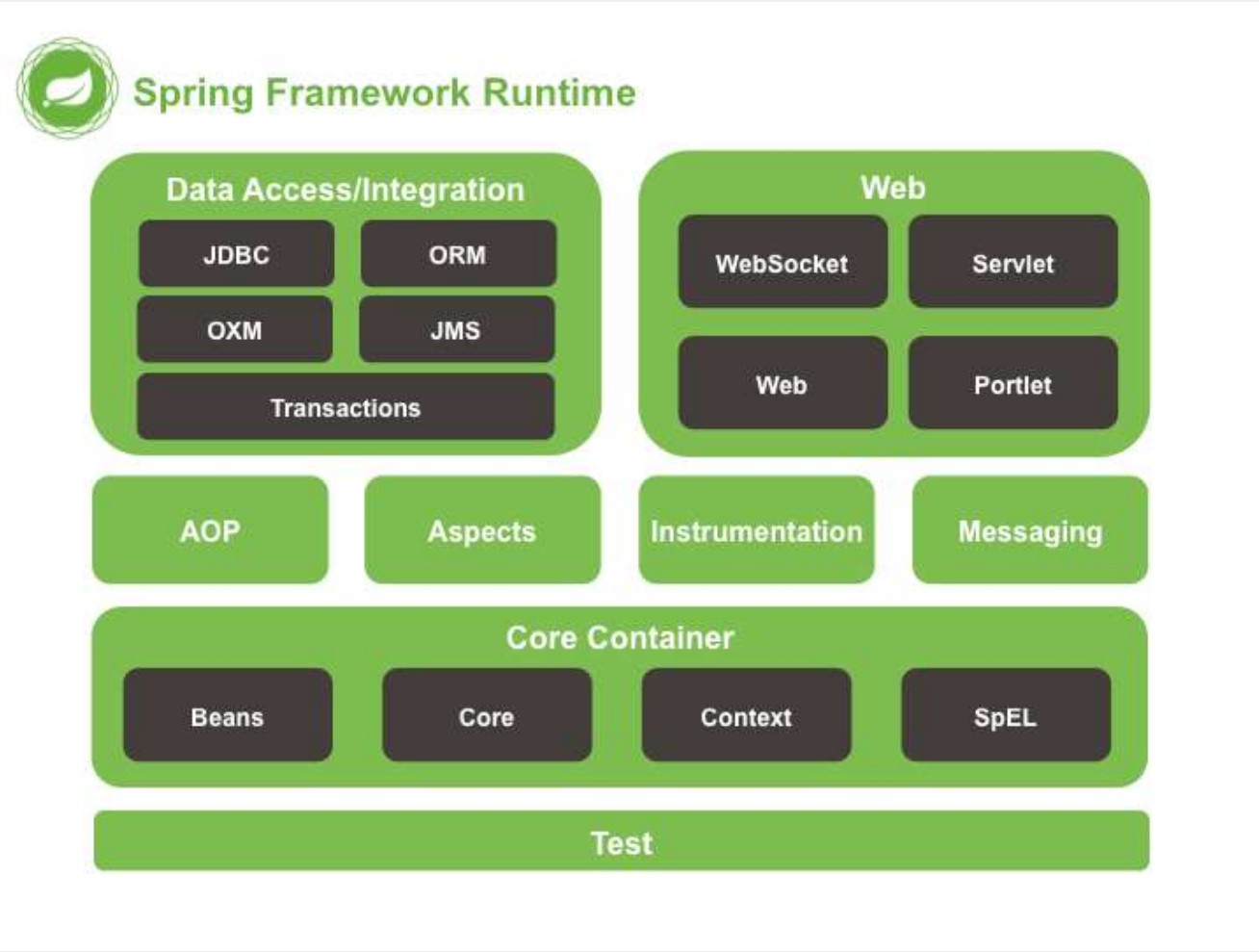
- Spring is the most popular application development framework for enterprise Java.
- Millions of developers around the world use Spring Framework to create high performing, easily testable, reusable code.
- The core features of the Spring Framework can be used in developing any Java application. It can be described as complete and modular framework. The Spring Framework can be used for all layer implementations of a real time application.
- Open source Java platform since 2003.
- Current version: 5.2

Spring

Feature

- It's **Lightweight**, in terms of both size (few MB) and overhead (negligible)
- It brings the principle of **Inversion of Control (IoC)**: custom-written portions of a computer program receive the flow of control from a generic framework (as opposed to traditional programming with reusable libraries)
- It supports **Dependency Injection (DI)**, objects are passively given their dependencies instead of creating or looking for dependent objects for themselves
- It's a **Container,** which contains and manages the lifecycle and configuration of application objects
- It's a **Framework,** which allows to configure and compose complex applications from simpler components

Spring's modular architecture is mainly composed of those levels



The Core package is the most important component of the Spring Framework.
This component provides the Dependency Injection features. The BeanFactory provides a factory pattern which separates the dependencies like initialization, creation and access of the objects from your actual program logic.

- The **Beans** module provides BeanFactory which is a sophisticated implementation of the factory pattern.
- The **Core** module provides the fundamental parts of the framework, including the IoC and Dependency Injection features.
- The **Context** module builds on the solid base provided by the Core and Beans modules and it is a medium to access any objects defined and configured. The **ApplicationContext** interface is the focal point of the Context module. **spring-context-support** provides support for integrating common third-party libraries into a Spring application context for caching (EhCache, Guava, JCache), mailing (JavaMail), scheduling (CommonJ, Quartz) and template engines (FreeMarker, JasperReports, Velocity).
- The **Expression Language** module provides a powerful expression language for querying and manipulating an object graph at runtime.

Spring

Architecture – AOP, Aspects, Instrumentation, Messaging, Test

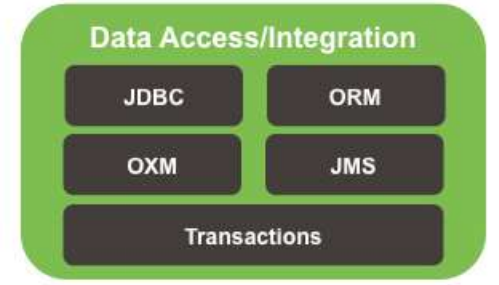
1. The spring-aop module provides an AOP Alliance-compliant aspect-oriented programming implementation. AOP is a programming paradigm that aims to increase modularity by allowing the separation of cross-cutting concerns. An aspect is a modularization of a concern that cuts across multiple classes.

2. The separate spring-aspects module provides integration with AspectJ.

3. The spring-instrumentation module provides class instrumentation support and classloader implementations to be used in certain application servers.

4. Spring Framework includes a spring-messaging module with key abstractions from the Spring Integration project such as Message, MessageChannel, MessageHandler, and others to serve as a foundation for messaging-based applications.

The spring-test module supports the unit testing and integration testing of Spring components with JUnit or TestNG. It provides consistent loading of Spring ApplicationContexts and caching of those contexts. It also provides mock objects that you can use to test your code in isolation.



The Data Access/Integration layer consists of the JDBC, ORM, OXM, JMS, and Transaction modules.

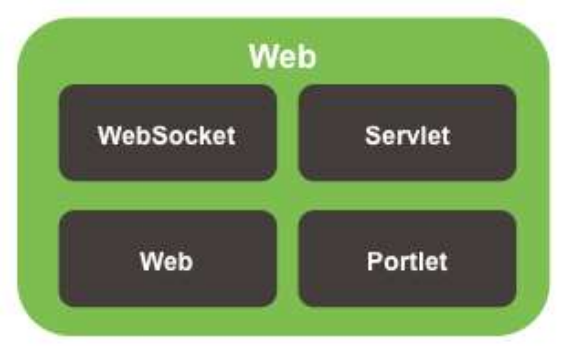
The spring-jdbc module provides a JDBC-abstraction layer that removes the need to do tedious JDBC coding and parsing of database-vendor specific error codes.

The spring-tx module supports programmatic and declarative transaction management for classes that implement special interfaces and for all your POJOs (Plain Old Java Objects).

The spring-orm module provides integration layers for popular object-relational mapping APIs, including JPA, JDO, and Hibernate. Using the spring-orm module you can use all of these O/R-mapping frameworks in combination with all of the other features Spring offers, such as the simple declarative transaction management feature mentioned previously.

The spring-oxm module provides an abstraction layer that supports Object/XML mapping implementations such as JAXB, Castor, XMLBeans, JiBX and XStream.

The spring-jms module (Java Messaging Service) contains features for producing and consuming messages. Since Spring Framework 4.1, it provides integration with the spring-messaging module.



The Web layer consists of the spring-web, spring-webmvc, spring-websocket, and spring-webmvc-portlet modules.

The spring-web module provides basic web-oriented integration features such as multipart file upload functionality and the initialization of the IoC container using Servlet listeners and a web-oriented application context. It also contains an HTTP client and the web-related parts of Spring's remoting support.

The spring-webmvc module (also known as the Web-Servlet module) contains Spring's model-view-controller (MVC) and REST Web Services implementation for web applications. Spring's MVC framework provides a clean separation between domain model code and web forms and integrates with all of the other features of the Spring Framework.

The spring-webmvc-portlet module (also known as the Web-Portlet module) provides the MVC implementation to be used in a Portlet environment and mirrors the functionality of the Servlet-based spring-webmvc module.

Spring

IoC Containers

The Spring IoC container is at the core of the Spring Framework. The container will create the objects, wire them together, configure them, and manage their complete life cycle from creation till destruction. The Spring container uses dependency injection (DI) to manage the components that make up an application. Spring provides following two types of containers.

BeanFactory container:

- It is an interface defined in org.springframework.beans.factory.BeanFactory.
- Bean Factory provides the basic support for Dependency Injection.
- It is based on factory design pattern which creates the beans of any type.
- BeanFactory follows lazy-initialization technique which means beans are loaded as soon as bean factory instance is created but the beans are created only when getBean() method is called.

BeanFactory beanFactory = new XmlBeanFactory(new ClassPathResource("beans.xml"));

ApplicationContext container:

- It is an interface defined in org.springframework.context.ApplicationContext.
- It is the advanced Spring container and it is built on top of the BeanFactory interface.
- ApplicationContext supports the features supported by Bean Factory but also provides some additional functionalities.
- ApplicationContext follows eager-initialization technique which means instance of beans are created as soon as you create the instance of Application context.
- Imp: *ClassPathXmlApplicationContext - FileSystemXmlApplicationContext*

ApplicationContext context=new ClassPathXmlApplicationContext("beans.xml");

"don’t call me, I’ll call you“

- Inversion of Control is a principle in software engineering by which the control of objects or portions of a program is transferred to a container or framework. It's most often used in the context of object-oriented programming.
- By contrast with traditional programming, in which our custom code makes calls to a library, IoC enables a framework to take control of the flow of a program and make calls to our custom code
- The advantages of this architecture are:
  - decoupling the execution of a task from its implementation
  - making it easier to switch between different implementations
  - greater modularity of a program
  - greater ease in testing a program by isolating a component or mocking its dependencies and allowing components to communicate through contracts
- Inversion of Control can be achieved through various mechanisms such as: Strategy design pattern, Service Locator pattern, Factory pattern, and Dependency Injection (DI).

cook
VS
delivery

Spring

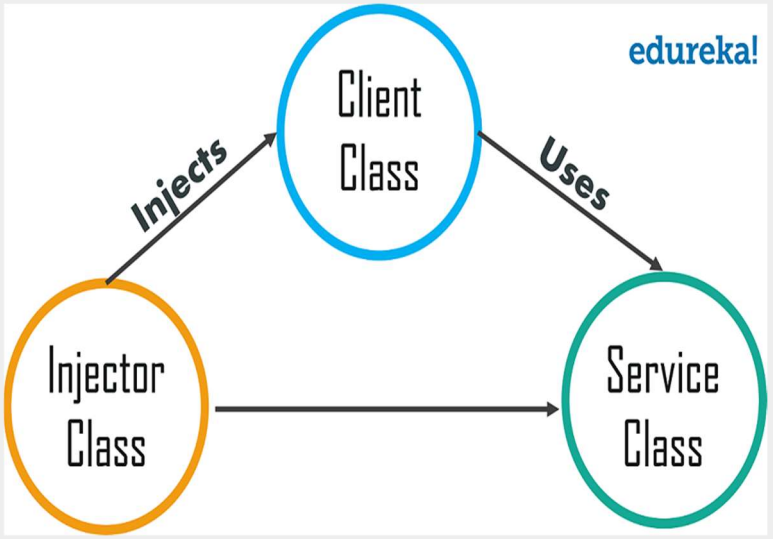
Dependency Injection

- Dependency in programming is an approach where a class uses specific functionalities of another class.
- The process of creating an object for another class, by launching the class to use the dependency directly, is called Dependency Injection.

- **Client Class**: it has a dependency of Service Class.
- **Service Class**: This class provides a service to the Client Class.

It's needed to create an instance of ServiceClass before to instantiate a ClientClass.

- **Injector Class**: This class is responsible for injecting the service class object into the client class



- Dependency Injection is also one of the core concepts of Spring Framework
- It is a design pattern that removes the dependency from the code.
- Spring Framework provides the dependencies of the class itself so that it can be easy to manage and test the application.
- Dependency Injection (or sometime called wiring) helps in gluing these classes together and at the same time keeping them independent.

```
public class TextEditor {
   private SpellChecker spellChecker;

   public TextEditor() {
      spellChecker = new SpellChecker();
   }
}
```

```
public class TextEditor {
   private SpellChecker spellChecker;

   public TextEditor(SpellChecker spellChecker) {
      this.spellChecker = spellChecker;
   }
}
```

Here, the TextEditor should not worry about SpellChecker implementation. The SpellChecker will be implemented independently and will be provided to the TextEditor at the time of TextEditor instantiation. This entire procedure is controlled by the Spring Framework.

Spring

Dependency Injection

The basic principle behind Dependency Injection (DI) is that objects define their dependencies only through constructor arguments, arguments to a factory method, or properties which are set on the object instance after it has been constructed or returned from a factory method. Then, it is the job of the container to actually inject those dependencies when it creates the bean. This is fundamentally the inverse, hence the name Inversion of Control (IoC).

TestSetterDI.java

```
public class TestSetterDI {

DemoBean demoBean = null;

public void setDemoBean(DemoBean demoBean) {
    this.demoBean = demoBean;
}
}
```

ConstructorDI.java

```
public class ConstructorDI {

DemoBean demoBean = null;

public TestSetterDI (DemoBean demoBean) {
    this.demoBean = demoBean;
}
}
```

In this methodology we implement an interface from the IOC framework. IOC framework will use the interface method to inject the object in the main class. It is much more appropriate to use this approach when you need to have some logic that is not applicable to place in a property. Such as logging support.

```
public void SetLogger(ILogger logger)
{
  _notificationService.SetLogger(logger);
  _productService.SetLogger(logger);
}
```

Exercise: 01-springbase

When the application starts, Spring will scan the components. All classes noted as @Component will be instantiated and registered in the ApplicationContext (the interface that represents the IoC container). The instances created will then be available to be injected into other beans, based on the dependencies declared.

Spring includes several annotations useful to define the components of a project. The main ones are:

- @Component: identifies a component that is a candidate for auto-detection
- @Service: Specialization of @Component, manage the business logic. It processes the data to provide it to the Controllers and vice versa sends the information to be saved to the data access layer.
- @Repository: Specialization of @Component, denotes classes that handle data access and manipulation (e.g. DAO classes)
- @Controller: Specialization of @Component, denotes classes that implement the controller layer of the app. Manages interactions between users and business logic. (e.g. web controllers)

- @Bean: the annotation indicates that the annotated method produces a bean to be managed by the Spring container.

  Spring @Bean Annotation is applied on a method to specify that it returns a bean to be managed by Spring context. Spring Bean annotation is usually declared in Configuration classes methods. In this case, bean methods may reference other @Bean methods in the same class by calling them directly.

  To declare a bean, simply annotate a method with the @Bean annotation. When JavaConfig encounters such a method, it will execute that method and register the return value as a bean within a BeanFactory. By default, the bean name will be that of the method name (see bean naming for details on how to customize this behavior).
- @Configuration: Annotating a class with the @Configuration annotation indicates that the class will be used by JavaConfig as a source of bean definitions. An application may make use of just one @Configuration-annotated class, or many

```
@Configuration
public class AppConfig {
    @Bean
    public TransferService transferService() {
        return new TransferServiceImpl();
    }
}
```

Scope	Description
singleton	This scopes the bean definition to a single instance per Spring IoC container (default).
prototype	This scopes a single bean definition to have any number of object instances.
request *	This scopes a bean definition to an HTTP request.
session *	This scopes a bean definition to an HTTP session.
global-session *	This scopes a bean definition to a global HTTP session.

https://howtodoinjava.com/spring-core/spring-annotations/

Spring Bean Autowiring

- Autowiring feature of spring framework enables you to inject the object dependency implicitly. It internally uses setter or constructor injection.
- In Spring, you can use @Autowired annotation to auto-wire bean
- You can enable annotation-driven injection by using below spring configuration or XML-based configuration

```
@Configuration
@ComponentScan("guru.springframework.autowiringdemo")
public class AppConfig {}
```

```
<context:annotation-config />
```

- Advantage: It requires the less code because we don't need to write the code to inject the dependency explicitly.
- Autowiring can't be used to inject primitive and string values. It works with reference only.

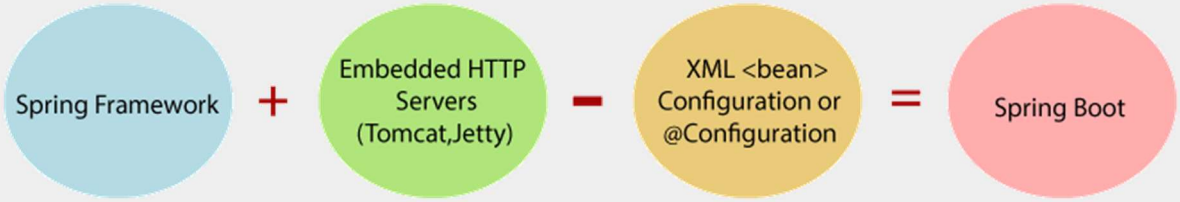
No	Mode	Description
1)	no	It is the default autowiring mode. It means no autowiring by default. (The default autowire mode in XML configuration)
2)	byName	The byName mode injects the object dependency according to name of the bean. In such case, property name and bean name must be same. It internally calls setter method.
3)	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. (The default autowire mode in java configuration)
4)	constructor	The constructor mode injects the dependency by calling the constructor of the class. It calls the constructor having large number of parameters.
5)	autodetect	It is deprecated since Spring 3.

Exercise: 02-springbase & 03-springbase

Spring

Spring Boot

- Spring Boot is a project that is built on the top of the Spring Framework. It provides an easier and faster way to set up, configure, and run both simple and web-based applications.
- It is a Spring module that provides the RAD (Rapid Application Development) feature to the Spring Framework. It is used to create a stand-alone Spring-based application that you can just run because it needs minimal Spring configuration.



Advantages

- It creates stand-alone Spring applications that can be started using Java -jar.
- It provides production-ready features such as metrics, health checks, and externalized configuration.
- There is no requirement for XML configuration.
- It increases productivity and reduces development time.
- It simplifies integration with other Java frameworks like JPA/Hibernate, Struts, etc.
- Spring boot helps in **resolving dependency conflict**. It identifies required dependencies and import them for you.
- It has information of **compatible version** for all dependencies. It minimizes the runtime **classloader** issues.
- It's "defaults configuration" approach, helps you in configuring most important pieces behind the scene. Override them only when you need. Otherwise everything just works, perfectly. It helps in avoiding **boilerplate code**, annotations and XML configurations.
- It provides embedded HTTP server Tomcat so that you can develop and test quickly.
- It has excellent integration with IDEs like eclipse and **intelliJ idea**.

Spring Initializr is a web-based tool provided by the Pivotal Web Service. With the help of Spring Initializr, we can easily generate the structure of the Spring Boot Project.

Initializr generates the project, which can be downloaded to your computer

- Project: It defines the kind of project. We can create either Maven Project or Gradle Project.
- Language: Spring Initializr provides the choice among three languages Java, Kotlin, and Groovy.
- Spring Boot: We can select the Spring Boot version. The latest version is 2.2.2.
- Project Metadata: It contains information related to the project, such as Group, Artifact, etc. Group denotes the package name; Artifact denotes the Application name. The default Group name is com.example, and the default Artifact name is demo.
- Dependencies: Dependencies are the collection of artifacts that we can add to our project.

To **run the application**, we need to use @SpringBootApplication annotation. Behind the scenes, that's equivalent to @Configuration, @EnableAutoConfiguration, and @ComponentScan together.

It enables the scanning of config classes, files and load them into spring context. In example below, execution start with main() method. It start loading all the config files, configure them and bootstrap the application based on application properties in application.properties file in /resources folder.

MyApplication.java

```java
@SpringBootApplication
public class MyApplication
{
    public static void main(String[] args)
    {
        SpringApplication.run(Application.class, args);
    }
}
```

application.properties

```
### Server port #########
server.port=8080

### Context root ########
server.contextPath=/home
```

Console

```
$ java -jar spring-boot-demo.jar
```

Exercise: 04-springboot

Web Services

Web Services are a mean to implement web APIs

- provide a service that is accessible through the HTTP protocol
- web service oriented architecture typically have a good separation of concern
- stateless web services scale well: knowing which specific instance is serving the request is not important then more instances can be spawned at any time without affecting sessions
- sometimes are improperly called REST services or RESTful services, but they should adhere to the REST architectural principle to be called so, which is often not the case