Spring

What is spring?
Its a container & a framework both.

Spring is an open source framework since February 2003.
Created by Rod Johnson.(currently hosted on pivotal/vmware)

One line answer to what is spring--- spring is not a J2EE specification BUT its created to make developing complex J2EE applications easier.

Why learn one more frmwork --- when u already have EJB,Struts,Hibernate etc....
Spring helps you to

1.Build applications from plain old Java objects (POJOs) (known as spring beans)
2. Apply enterprise services non-invasively.
(w/o invasion means --- POJOs DONT implement or extend from spring APIs) This capability applies to the Java SE programming model and to full and partial Java EE.

Examples of how you, as an application developer, can use the Spring platform advantage:

Make a Java method execute in a database transaction without having to deal with transaction APIs.

Make a local Java method a remote procedure without having to deal with remote APIs.

Make a local Java method an ORM operation without having to deal with overheads of ORM set up.

Make a local Java method a web service end point , without having to deal with JAX WS or JAX RS setups.

Simple answer to WHY Spring

Spring simplifies Java development.
Since above is a bold stmt -- to justify it --- there are main 4 reasons

Reasons ---it applies 4 key strategies
1. lightweight & min intrusive(POJOs not tied to spring) development with POJOs

2. Loose coupling thro' DI/IoC & with usage of i/f

3.Declarative prog(XML or anno or java config) + thro' Aspects & common conventions

4. Boilerplate code reduction thro aspects & templates.

7. Def. of Spring ----
Spring is a lightweight , dependency injection and aspect-oriented container and framework. works on Ioc(Inversion of control)

Meaning

7.1 Lightweight Lesser no of JARs
JavaBeans / POJOs in Spring-enabled application often have no dependencies on Spring-specific classes.

7.2 Dependency Injection Spring allows loose coupling through dependency injection (DI).

DI =instead of an object looking up dependencies from a container(in EJB from EJB container or in RMI from RMIRegistry) or creating its own dependency(in Fixed JDBC , conn = DM.getCn(....)) , the container gives the dependencies to the object at instantiation without waiting to be asked.

You can think of D.I as JNDI(Java naming & directory iterface -- Naming service) in reverse.

7.3 Aspect-oriented Spring supports aspect-oriented programming(AOP)

(AOP) allows separating application business logic from system services (such as auditing and transaction management,logging , security,transactions).
Application objects perform only business logic and nothing more.

They are not responsible for (or even aware of) other system concerns, such as eg : logging , transactions, or security

---

7.4 Why Spring is a Container ?

Spring is a container which manages
the lifecycle and configuration of application objects.(spring beans)
In Spring, using config XMLs or annotations or java config, you can
declare - how to create each of your application objects(spring beans)
- how to configure them
- how they should be associated with each
other.(collaboration/wiring/coupling=connecting dependencies with dependent objs)

7.5 Why Spring is a framework ?

Spring allows you to configure and compose complex applications from simpler components. In Spring, application objects are composed declaratively, typically in an XML file or using annotations

Spring also provides you with ready made implemetations of services like - transaction management, persistence framework,web mvc etc.

It provides readymade implementation of patterns & helps in building enterprise

applications.

It is the most popular application development framework for enterprise Java. It is used to create high performant, easily testable, and reusable code.

Spring framework is an open source Java platform.
Founder is Rod Johnson and was first released under the Apache license in June 2003.
Currently hosted on Pivotal/VMware

Spring is unique, for several reasons:

1. It helps you in important areas that many other popular frameworks don't. eg : readymade Hibernate templates.

2. Provides a way to manage your business objects - based on Dependeny injection(DI)

3.Spring is both comprehensive and modular. Offers you lot many features, yet gives you choice to integrate layers one by one, test it & then add new features via new layers.

4 Spring is an ideal framework for test driven projects.

5. It is basically a one-stop shop, addressing most of the concerns of typical enterprise applications.

6.Using Spring one can centrally describe collaborating objects. From the earliest versions of Spring, there was an XML file that was used to describe the object graph.
Contents of XML ---- Consists of beans. Each bean element describes an object that will be created and given an id. Each property element describes a setter method on the object and the value that should be given to it. These setters are called for you by the Spring application container.

Main winning feature of Spring is : loose coupling between the modules.

How does it achieve loose coupling ?
1. IoC -- IoC is achieved using Dependency Injection(D.I)
2. Aspect Oriented Programming(AOP)

What is Spring ?

1. An open source framework since February 2003.
Created by Rod Johnson and described in his book Expert One-on-One: J2EE Design and Development.

Allows us to give capability of EJBs to plain JavaBeans without using an application server.

Any Java SE application can also use Spring to get simplicity, testability, and loose coupling.

2. Spring has been hosted on SourceForge.

3. Spring is a lightweight framework. Most of your Java classes will have no dependency on Spring. This means that you can easily transition your application from the Spring framework to any other frmwork. (Framework independence)

4. All Java applications that consist of multiple classes have inter-dependencies or coupling between classes. Spring helps us develop applications that minimize the negative effects of coupling and encourages the use of interfaces in application development.

5. Using interfaces in our applications to specify type helps make our applications easier to maintain and enhance later.

6. The Spring framework helps developers for separation of responsibilities.

eg scenario --
Think of a situation Your manager tells you to do your normal development work(eg - write stock trading appln) + write down everything you do and how long it takes you.

A better situation would be you do your normal work, but another person observes what you re doing and records it and measures how long it took.

Even better would be if you were totally unaware of that other person and that other person was able to also observe and record , not just yours but any other people s work and time.

That s separation of responsibilities. --- This is what spring offers u thro AOP

8 Spring framework Modules

9. Advantages of ApplicationContext over BeanFactory
9.1 Application contexts resolve text messages, including support for internationalization (I18N).
9.2 Application contexts provide a generic way to load file resources, such as images.
9.3 Application contexts can publish events to beans that are registered as listeners.

IOC -- rather a generic term
Inversion of Control (IoC) is an object-oriented programming practice where the object coupling(dependent obj bound with dependency) is bound at run time by an

assembler object(eg spring container) and is typically not known at compile time using static analysis.

Unlike in traditional prog -- where dependent obj creates dependencies leading to tight coupling , container sets the dependencies (not US --not a prog or not a dependent obj) ---so its inversion of control

Instead of dependent objs managing their dependencies , the control of managing the dependencies lies with SC
eg : UserController :
dep : @AutoWired
private UserService service; //field level D.I

Dependency. injection=Ioc+dependency inversion
Why IoC or Dependency Injection (advantages of IoC)
    * There is a decoupling of the execution of a certain task from implementation.
    * Every module can focus on what it is designed for.
    * Modules make no assumptions about what other systems do but rely on their contracts/specs (=i/f)
    * Replacing modules has no side effect on other modules.

Inversion of Control is sometimes referred to as the "Hollywood Principle: Don't call us, we'll call you",
---

What is dependency injection ?
In JSP---JB---DAO(Utils) -- POJO --DB layers
Dependent Objs -- JavaBean , Hibernate based DAO, JDBC Based DAO
Dependencies --- DAO,HibUtils(SessionFactory) , DBUtils(DB connection)

All of above are examples of tight coupling.

Why --Any time the nature of the dependency changes , dependent obj is affected(i.e u will have to make changes in dependent obj)
eg : When the dependency of Java Bean changes from JDBC Based DAO to Hibernate based DAO , in case of user authentication , javabean class has to be modified to handle invalid login case(i.e handle NoResultException)

Tight coupling --strongly un desirable.
Why -- difficult to maintain or extend.

In above examples , Java bean creates the instance of DAO.
Hibernate based DAO , gets SF from HibUtils.
JDBC based DAO ,gets db connection from DBUtils.

i.e dependent objects are managing their dependencies. ---traditional/conventional programming model.

Dependency injection (DI) is the ability to inject dependencies. DI can help make your code architecturally pure. It aids in using a design by interface approach as well as test driven development by providing a consistent way to inject dependencies. For example a data access object (DAO) may need a database connection. Thus the DAO depends on the database connection. Instead of looking up the database connection with JNDI, you could inject it. or another eg is JMS -- conn factory or destination

One way to think about a DI container like Spring is to think of JNDI turned inside out. Instead of the objects looking up other objects that it needs to get its job done (dependencies), with DI the container injects those dependent objects. This is the so-called Hollywood principle, you don't call us (lookup objects), we will call you (inject objects).

What is D.I ?(Dependency injection=wiring=collaboration between dependent & dependency)
Instead of dependent objs managing their dependencies , 3rd party containers(eg : Angular / Spring/ EJB/ WC) will auto create the dependecies & make it available to dependents, directly @ run time.

Since dependent are no longer managing dependencies --its called as IoC ---Inversion of control

Hollywood principle --You don't call us , we will call you....
SC --- > Dependent objs (i.e SC will create the dependencies for the dependent objs)

eg : UserController
```
@Autowired
private IUserService service;
```

In DAO layer
```
@AutoWired
private SessionFactory sf;
```

More details about <bean> tag
Attributes
1. id --mandatory --bean unique id
2. class --- mandatory -- Fully qualified bean class name
3. scope --- In Java SE --- singleton | prototype
   In web app singleton | prototype | request | session | global session
   Default scope = singleton
   singleton --- SC will share single bean instance for multiple requests/demands (via ctx.getBean)
   prototype -- SC creates NEW bean instance per request/demand.
4. lazy-init --- boolean attribute. default value=false.
   Applicable only to singleton beans.

SC will auto create singleton spring bean instance --- @ SC start up.
5. init-method --name of init style method(public void anyName() throws Exception {..})
   called by SC after setter based D.I
6. destroy-method --name of destroy style method
   (public void anyName() throws Exception{..})
   called by SC before GC of spring bean (applicable only to singleton beans)

More on ApplicationContext

The instantiation of the ApplicationContext creates the container that consists of the objects defined in that XML file.
The purpose of this XML file is to create the beans and their relationship.
This XML file is then provided to the ApplicationContext instance, which creates a container with these beans and their object graphs along with relationships. The Spring container is simply a holder of the bean instances that were created from the XML file.
An API (getBean) is provided to query these beans and use them accordingly from our client application.

More on Ioc

IoC is also known as dependency injection (DI).

It is a process whereby objects define their dependencies,that is, the other objects they work with, only through constructor arguments, arguments to a factory method, or properties that are set on the object instance after it is constructed or returned from a factory method.

The container then injects those dependencies when it creates the bean.

This process is fundamentally the inverse,(hence the name Inversion of Control (IoC)), of the bean itself controlling the instantiation or location of its dependencies by using direct construction of classes, or a mechanism such as the Service Locator pattern.

The org.springframework.beans and org.springframework.context packages are the basis for Spring Framework's IoC container. The BeanFactory interface provides an advanced
configuration mechanism capable of managing any type of object. ApplicationContext is a
sub-interface of BeanFactory. It adds easier integration with Spring's AOP features; message resource handling (for use in internationalization), event publication; and application-layer specific contexts such as the WebApplicationContext for use in web applications.

Spring Framework Modules

1. Core Container
It consists of the Core, Beans, Context, and Expression Language modules

1.1 The Core module provides the fundamental parts of the framework, including the IoC based upon Dependency Injection

1.2 The Bean module provides BeanFactory , which is a readymade implementation of the factory pattern.

1.3 The Context module is based upon the Core and Beans modules .It provides a way to access any spring beans(objects managed by Spring container) which are defined and configured. The ApplicationContext interface is the heart of Context module.

1.4 The SpEL module provides a powerful expression language for querying and manipulating an object graph at runtime.

---

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

2.1 The JDBC module provides a JDBC-abstraction layer that removes the need for boilerplat JDBC related coding.

2.2 The ORM module provides integration layers for popular object-relational mapping APIs, including JPA, JDO, Hibernate, and iBatis.

2.3 The OXM module provides an abstraction layer that supports Object/XML mapping implementations for JAXB, XMLBeans, , XStream etc

2.4 JMS module contains features for producing and consuming messages sent across application components

2.5 The Transaction module supports programmatic and declarative transaction management for typically service layer or DAO layer spring beans.

---

3. Web
The Web layer consists of the Web, Web-MVC, Web-Socket, and Web-Portlet modules .

The Web module provides basic web integration features such as MVC support,multipart file-upload functionality ,init of IoC container using servlet listeners , web application context , I18N etc.

3.1 The Web-MVC module contains Spring's Model-View-Controller (MVC) implementation for web applications.

3.2 The Web-Socket module provides support for WebSocket-based, two-way communication between the client and the server in web applications.

3.3 The Web-Portlet module provides the MVC implementation to be used in a portlet environment

---

Other important Modules

1. The AOP module provides an aspect-oriented programming implementation allowing you to define method-interceptors and pointcuts to cleanly decouple code that implements functionality that should be separated.

2. The Instrumentation module

3. The Messaging module provides support for STOMP as the WebSocket sub-protocol to use in web socket applications.

4. The Test module supports the testing of Spring components with JUnit or TestNG frameworks.

XML based config

What are 3 different ways of supplying metadata instrs to SC
1. pure xml (legacy!)
2. Hybrid approach (lesser xml + majority annotations)
3. Java config class + annotations

1.Pure XML based approach(legacy)
<bean> tag attributes
id : unique bean id
class : F.Q bean class name
scope :
Java SE : singleton | prototype
def scope : singleton (=> SC creates a single instance of this bean : to be shared across multiple demands i.e getBean)
prototype : => SC creates a separate instance of the bean : as per demand i.e getBean)
lazy-init : def value =false (applicable to ONLY singleton beans)

init-method : name of custom init method
pattern : public void anyName() throws Exc {....}
Will be invoked for singleton as well as prototype beans

destroy-method : name of custom destroy method : SC invokes it just before GC
pattern : public void anyName() throws Exc {....}
Will be invoked only for singleton beans

How to launch SC in a standalone application ?

Understand SC API (refer to a diagram)
API
How to get ready to use spring beans from SC ?
API of BeanFactory
public <T> T getBean(String beanId,Class<T> beanClass) throws BeansException
T : type of the spring bean

Different Modes of Wiring (D.I.) : (refer to a diagram)

Hybrid approach(Reduced xml, reduced java code + majority annotations)

Steps in Spring programming using annotations

0. To enable annotation support -- add context namespace & add the following
<context:annotation-config/> --- To tell SC --to enable annotation support(eg ---
AutoWired,PostConstruct,Predestroy,.....)

0.5 --- How to specify location(base pkg) of spring beans to SC?
<context:component-scan base-package="comma sep list of pkgs"/>---
SC starts searching(scanning) in specified pkgs (including sub-pkgs) ---for
classes anno with stereotype annotations ---
@Component,@Service,@Repository,@Controller,@RestController,@ControllerAdvice...

Basic class level annotations meant for SC
Super type
@Component --- spring bean class
sub - type annotations
@Controller --- In Web MVC scenario -- for request handling.
@Service --- Service layer (B.L) + transaction management
@Repository --- DAO layer
@RestController -- RESTful service provider
---
1. @Component --- <bean id , class....> --- SC interprets it & starts bean
life-cycle.
eg ---
package beans;
@Component("abc")
public class MyBean {...}
xml --- <bean id="abc" class="beans.MyBean"/>
OR
@Component
public class MyBean {...}
xml --- <bean id="myBean" class="beans.MyBean"/>

2. @Controller -- spring web mvc controller

3. @Repository --- DAO layer class

4. @Service --- for service layer beans --- transactions.

5. @Scope(value="singleton|prototype|request|session")--- class level annotaion --- in xml via scope attribute.

6. @Lazy(true|false) ----class level anno -- lazy-init attribute

7. @PostConstruct ---method level anno - init-method ---method level

8. @PreDestroy ---method level anno-- destroy-method --- method level

9. @Required(true|false) --- setter method or paramed constr ---tells SC if depenency is mandatory or optional-- def=true

10. @AutoWired ---setter method or paramed constr or field level
eg --- TestTransport implements Transport {...}
 autowire="byType"
eg -- field level annotation ---in ATMImpl bean (dependent)
@AutoWired //autowire=byType , mandatory by default (required=true)
private Transport myTransport;
Meaning -- no parameterised constr, no setter , no xml containing bean definition is required.
SC --- chks for any bean of Transport by type & injects it in ATMImpl
What if : SC comes across multiple matches : SC throws NoUniqueBeanDefinitionException
What if : SC doesn't find even a single match : SC throws UnSatisfiedDependencyException

11. @AutoWired//(required=true)
@Qualifier("test")
private Transport myTransport; ---- =>autowire="byName"
---spring supplied anno.
SC searches for a bean with id="test"
Match found : NO excs , field level D.I succeds!
Match not found : SC throws exception (UnSatisfiedDependencyException : NoBeanDefFoundExc)
OR
@Resource(name="soap")
private Transport myTransport; ---- autowire="byName"

--J2EE supplied via javax.annotation
