Why Spring ?
Simplifies overall java development

What is it ?
container --manages life cycle of spring beans
(spring bean --- java obj whose life cycle completely managed by SC(spring container)
eg : rest controller, controller, service,DAO.
framework --rdy made implementation of std patterns(eg :MVC,Proxy,singleton,factory, ORM ...)
Spring is modular n extensive framework.

Why Spring : loosely coupled application
Via : D.I / AOP

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.

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)

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

Spring bean life cycle
Types of wiring
```