* IoC Container – este responsabil de crearea, configurarea si asamblarea beanurilor.
* IoC Container primeste toate datele necesare dintr-un fisier xml, desi nu e unica metoda
* Rolul la IoC Container:
* instantiaza application class
* configureaza obiectul
* asambleaza dependentele intre obiecte

**Tipuri de IoC Container**

* **BeanFactory**
* **ApplicationContext**

**Bean Factory IoC Container**

* Acesta este cel mai simplu tip de container care ofera suport de baza pentru Dependecy Injection si e definit de interfata

*org.springframework.beans.factory.BeanFactory*

* Sunt diferite implementatii ale aceste interfate, pentru a citi de ex date din XML, ca XmlBeanFactory

ClassPathResource resource=**new** ClassPathResource("applicationContext.xml");

BeanFactory factory=**new** XmlBeanFactory(resource);

* Este mai vechi, dar inca utilizata uneori. Totusi, este depreciata din Spring 3.0
* Este bun cand resursele sunt limitate, ca in mobile apps.

**ApplicationContext IoC Container**

* Este un advanced container, in varful la Bean Factory.
* Tot este o interfata ce are tot ce are si BeanFactory
* Spre deosebire de BeanFactory, ofera mai multe posibilitati, ca:
* MessageSource
* Acces la resources
* Comunicarea cu difertie contexts etc.
* Clase ce o impleneteaza:

**FileSystemXmlApplicationContext – preia metadatele dintr-un fisier XML extern, nu din proiect. Deci, nu se cauta nimic in classpath, dar are nevoie de full path al fisierului xml.**

**ClassPathXmlApplicationContext – fisier xml in webapp sau putem folosi si classpath:**

**WebXmlApplicationContext – ia datele de la un XmlBeanDefiniotReader. Este destinata unei web aplicatii.**

**AnnotationConfigApplicationContext –** ia datele de la o clasa

**O diferenta majora e ca BeanFactory container creaza beanurile in mod lazy, adica le creaza doar cand sunt cerute, dar ApplicatinContext creaza toate beanurile singleton la crearea sa chiar.**

**Despre FrontController Design Pattern**

* Front Controller este o componenta speciala din java applications care poate prelua requesturi si sa le trimita la componente ce sa le prelucreze, apoi sa ia raspunsul de la aceste componente si sa le trimita la View Component(ca JSP), apoi preia aceste rezultate formatate de View Controller si le trimite la browser.
* Front Controller deci controleaza intreaga activitate de pe o aplicatie WEB
* Necesitatea lui apare din faptul ca Clasele Java, care prelucreaza requesturile la user, nu le pot intercepta direct, de aceea apare necesitatea de a crea altele, ce sa se ocupe anume de asta.
* Anume de interceptarea requesturilor si trimiterea lor catre clasa, apoi a raspunsului catre View Controller si client(browser) se ocupa un Servlet.
* Deoarece Front Controller face atat de multe lucruri, cel mai bine e sa il cream chiar la rularea aplicatiei, deci sa cream toate obiectele sale, si anume asta o facem aici in web.xml:

<load-on-startup>1</load-on-startup>

Daca nu vom face asta, DispatcherServlet va fi initializat abea cand va veni un prim request, si noi nu vrem asta.

* In Spring, nu cream un intreg Front Controller de la 0, dar folosim unul deja existent, si anume asta si se numeste **DispatcherServlet**
* Anume la crearea lui DispatcherServlet, se creaza si IoC Controller de tip XmlWebApplicationContext, pe baza la configuratia .xml data de noi in ApplicationContext.xml
* Beanurile singleton sunt pastrate in cache la IoC Container
* La crearea lui Servlet, se creaza un Servlet Container, unde se pastreaza toate obiectele necesare pentru functionarea lui.

**Containters in Spring**

* Orice webapp in Spring va avea cel putin 2 containere. Unul e creat de ContextLoaderListener  aici:
* <listener>  
   <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  
  </listener>

altul va fi creat aici de DispatcherServlet

<servlet>  
 <servlet-name>dispatcher</servlet-name>  
 <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  
 <load-on-startup>1</load-on-startup>  
</servlet>

DispatcherServlet container e copil container a lui ContextLoaderListener

* Fiecare DispatcherServlet creat are un Container Separat
* Un container e creat fie cand noi cream un obiect de tip ApplicationContext(parinti ai lui ca Xml...,Annotation....) fie de dispatcher servlet, caci doar la crearea configuratiei pentru dispatcher serlvet, noi specificam si appcontexturile, mai multe chiar.
* O spring app poate avea oricate containere
* Context se refera mereu la un Container, deci sunt cam acelasi lucru in teorie.

Daca mai multe containere au un bean cu acelasi ID, si dipatcherServlet le va folosi pe ambele, cand vom da @Autowired, se va alege beanul la ultima applicationContext pusa:

protected Class<?>[] getServletConfigClasses() {  
 return new Class[] {AppContext1.class, AppContext2.class};  
}



**Deci, desi ambele au un bean identic, ca id si tip, la @Autowired se va alege beanul de la AppContext2.class, deoarece containerul e initializat dupa AppContext1**