ApplicationContext

- In a Spring-based application, the application objects live within the Spring Container.
  - The container creates the objects, wire them together, configure them, and manage their lifecycle.

ApplicationContext

- The *ApplicationContext* interface standardizes the Spring bean container behavior.
  - Its implementations form the simple container, providing basic support for DI.
  - Supports I18N.
  - Supports Event Management.

ApplicationContext

- Spring comes with several flavors of *ApplicationContext.*
  - *FileSystemXmlApplicationContext*
  - *ClassPathXmlApplicationContext*
  - *XmlWebApplicationContext*

FileSystemXmlApplicationContext

- Loads a context definition from an XML file located in the file system.
- E.g.

```
ApplicationContext ctx;
String file = "c:/config.xml";
ctx =
new
FileSystemXmlApplicationContext(file);
```

ClassPathXmlApplicationContext

- Loads a context definition from an XML file located in the classpath.
- E.g.

```
ApplicationContext ctx;
String file = "config.xml";
ctx =
new ClassPathXmlApplicationContext(file);
```

XmlWebApplicationContext

- Loads a context definition from an XML file contained within a web application.
- Used in Spring MVC environment.

Bean Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<beans>
   <bean id="emp"
   class="com.emp.Employee">
   </bean>
</beans>
```

Bean Configuration File

- **bean:** The Element which is the most basic configuration unit in Spring. It tells Spring Container to create an Object.
- **id**: The Attribute which gives the bean a unique name by which it can be accessed.
- **class**: The Attribute which tells Spring the type of a Bean.

Accessing Bean

```
ApplicationContext ctx;
String file = "c:/config.xml";
ctx = new
FileSystemXmlApplicationContext(file);

GreetingService service;
service =
(GreetingService)ctx.getBean("greet");
```

How DI/IoC Container works

- In Inversion Of Control (IoC), control is inverted back to the Container to support the object dependencies.
- IoC container creates the POJO objects and provides dependencies to them.
  - These POJO objects are not tied to any framework.
- The declarative configuration for POJO objects is defined with unique identities (id) in XML.
  - These are known as bean definitions.

How DI/IoC Container works

- The IoC container at runtime identifies POJO bean definitions, creates bean objects and returns them to the Application.
- IoC container manages dependencies of the objects.

Injecting the Dependencies

- The IoC container will Inject the dependencies in three ways
  - Create the POJO objects by using no-argument constructor and injecting the dependent properties by calling the setter methods.

Injecting the Dependencies

- The IoC container will Inject the dependencies in three ways
  - Create the POJO objects by using no-argument constructor and injecting the dependent properties by calling the setter methods.
  - Create the POJO objects by using parameterized constructors and injecting the dependent properties through the constructor.

Injecting the Dependencies

- The IoC container will Inject the dependencies in three ways
  - Create the POJO objects by using no-argument constructor and injecting the dependent properties by calling the setter methods.
  - Create the POJO objects by using parameterized constructors and injecting the dependent properties through the constructor.

Injecting Properties by calling Setters

```
public Class Employee {
   private String fname, lname;
  public Employee(){}

  public void setFname(String f) {
          fname = f;
  }
  public void setLname(String l) {
          lname = l;
   }
}
```

Injecting Properties by calling Setters

```
<bean id="emp"
class="com.emp.Employee">
  <property name="fname" value="a"/>
 <property name="lname" value="b"/>
</bean>
```

Dependent Beans

Dependent Beans

- A bean is a dependency of another bean, is expressed by the fact that, one bean is set as a property of another.

Dependent Beans

- It is achieved with the `<ref/>` element or `ref` attribute in XML based configuration metadata of beans.

Bean Loading

Bean Loading

- In Spring, Bean Loading happens by 2 ways:
  - EAGER (DEFAULT)
  - LAZY

Bean Loading

- The bean registered in the configuration unit gets instantiated as soon as the `ApplicationContext` is built.
- This is known as EAGER Loading.

Bean Loading

- The bean registered in the configuration unit gets instantiated only when the client program makes a request for the same.
- This is known as LAZY Loading.

Bean Scopes

- Every bean registered in XML file has some scope.
- It is possible to modify scope of the bean using `scope` attribute of <bean> element.

Bean Scopes

- There are 5 different types of scopes:
  - singleton
  - prototype
  - request
  - session
  - global-session

Bean Scopes

- singleton
  - It is the default scope.
  - Indicates that the bean configuration is singleton.
  - If the same bean is requested multiple times, spring returns the same object.

Bean Scopes

- prototype
  - Antonym of singleton.
  - If the same bean is requested multiple times, spring returns the a new object every time.

Bean Scopes

- request
  - Applicable only in the context of Spring MVC.
  - The bean is alive until the response is generated.
  - For every new instance of HttpServletRequest, spring creates a new instance.

Bean Scopes

- session
  - Applicable only in the context of Spring MVC.
  - The bean is alive until the session is over.
  - Bean can survive even if the response is generated.
  - For every new instance of HttpSession, spring creates a new instance.

Bean Scopes

- global-session
  - Applicable in the context of Spring Portlet environment.
  - The bean is alive across multiple portlets.