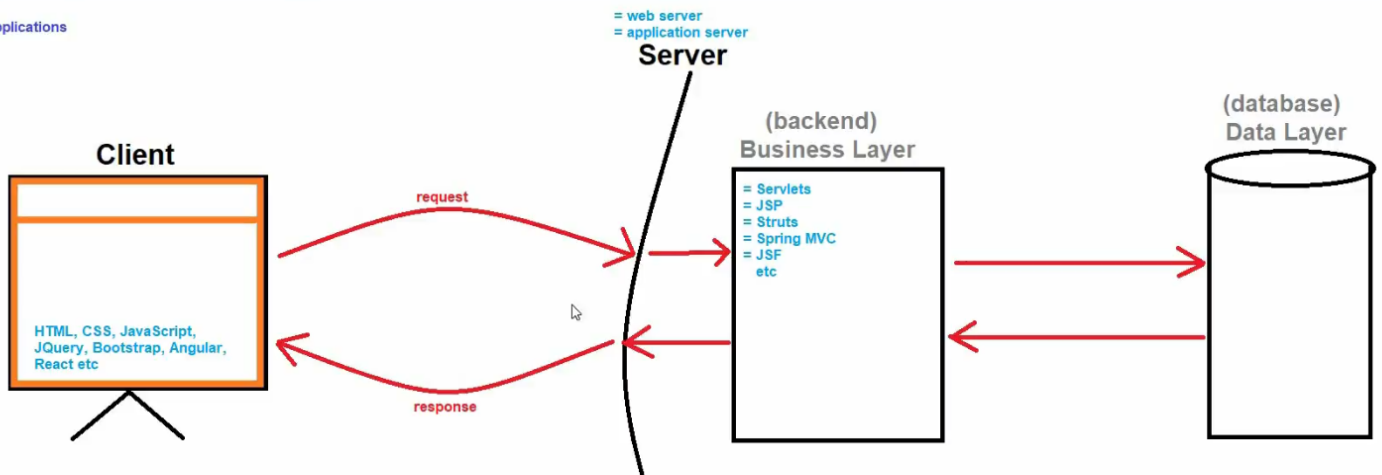
Architecture of Web Applications



=> Editions In Java :-

-> There are 3 editions in java :-

1. J2SE (Java 2 Standard Edition) - Core Java
2. J2EE (Java 2 Enterprise Edition) - Advance Java
3. J2ME (Java 2 Micro Edition) - Mobiles, Embded System (remotes, ATM's, TV, Washing Machines etc)

=> Types of applications in java :-

-> We can create 2 types of applications in java :-

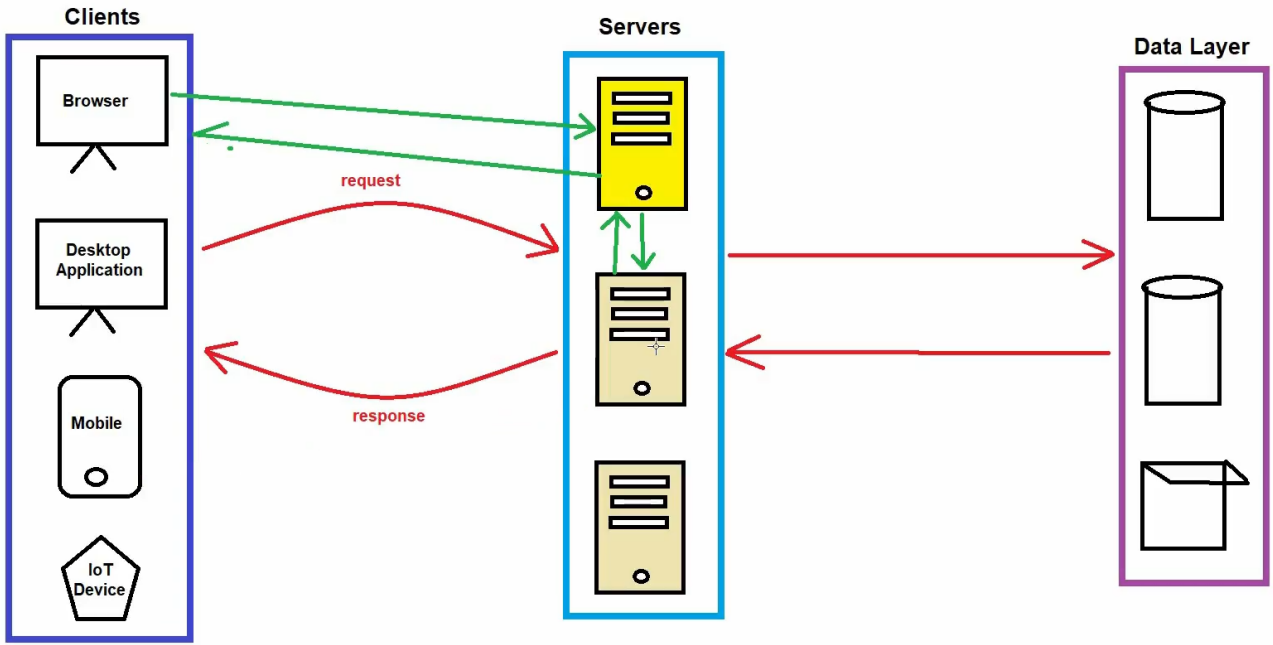
1. Standalone Applications
   - = These are the applications which are executed only on single system
   - = These applications can be developed using J2SE
   - = These applications does not follows the client-server architecture
   - = There are 2 types of standalone applications :-
     a. CUI (Character User Interface) Applications
        - Console Based App or Command Line User Interface or Text-Based App
     b. GUI (Graphical User Interface) Applications
2. Enterprise Applications

=> What is Enterprise ?

-> "Enterprise" term is used for large scale companies which has multiple departments, levels, divisons or groups

-> For eg :-

- = TATA Group : Consumer and retail, Hotels, IT, Automobiles, Steel, Power etc
- = Mahindra Group : IT, Automobiles, Defence, Education, Financial Services etc etc



=> What is enterprise applications ?

-> "Enterprise Applications" are large-scale, distributed, transaction and highly available applications which are designed to support the enterprise business requirements

-> To develop enterprise applications we have to use a lot of technologies, multiple design patterns, system architectures .

=> Web Applications :-

-> Client : Browser

-> Server : Web Server or Application Server

-> Technologies Used : Servlet, JSP, Spring MVC, JSF, Play Framework, Struts etc

-> Architecture :-

=> Distributed Applications :-

-> Client : Browser, Desktop Application, Mobile Application, IoT Device etc

-> Server : Application Server

-> Technologies Used : EJB (Enterprise Java Beans), Spring framework, JPA (Java Persistence API), Hibernate, JTA (Java Transaction API), JMS (Java Message Service) etc

-> Architecture :-

=> Difference between Web Server and Application Server :-

= Web Server is lightweight
  Application Server is heavy weight

= Web server contains only web containers (servlet container, JSP container etc)
  Application server contains web container + EJB container

= Web server is good for static contents like html pages
  Application server is good for dynamic contents

= Web server consumes less resources i.e. CPU, memory etc
  Application server use more resources

= Web server examples are : Apache Tomcat, Resin etc
  Application server examples are : Weblogic, JBoss, Websphere etc

=> What is framework ?

-> Frameworks are the pre-written code acting as a template which can be used or customized by the developers

-> In simple terms we can say that frameworks are the collections of API's and tools which can be used to develop projects

-> For example : Spring, Struts, Hibernate, Angular, React etc

=> Advantages of frameworks :-

-> Fast development speed

-> Less code (because it removes the boilerplate code)

-> Support API integration

-> Customizable (open source)

-> Easy to debug

-> Good documentation support

etc

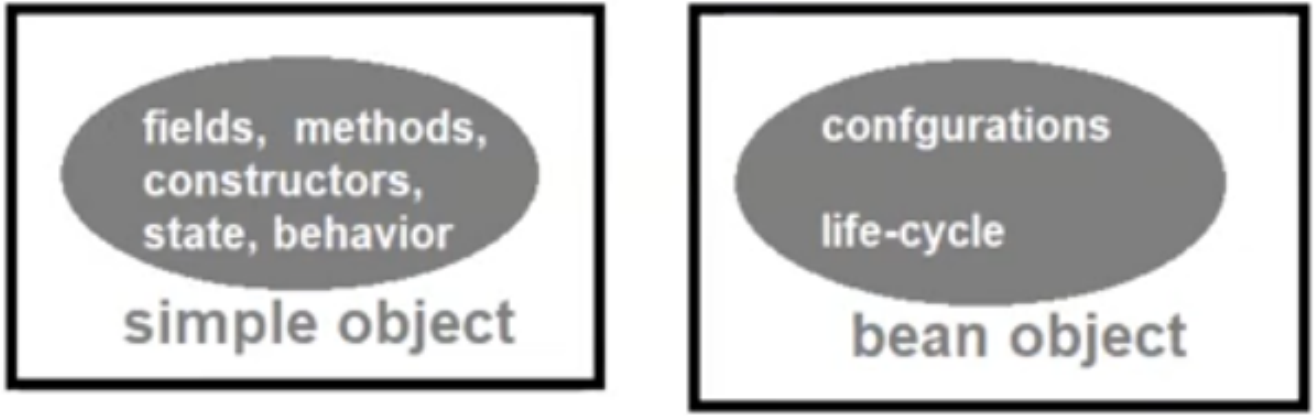
=> Types of frameworks :-

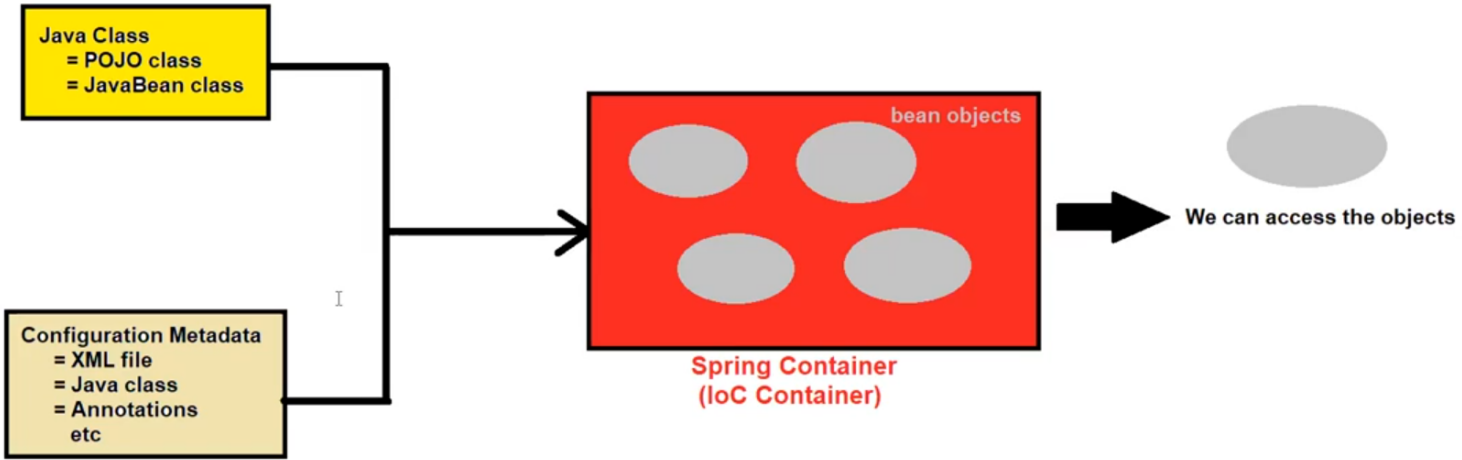
-> There are 2 types of frameworks :-

1. Web Framework
   = eg. Struts, JSF etc
2. Application Framework
   = eg. Spring etc

=> What is Spring ?
-> Spring is an "Open Source Application Framework" which is used to develop any type of application i.e. Standalone Application or Enterprise Application
-> Spring framework was written by Rod Johnson
-> Spring framework was released under Apache 1.0 licence
-> Spring framework was released in June 2003
-> First production version i.e. 1.0 version was released in March 2004
-> Latest version is 6.x version

=> Advantages of Spring Framework :-
1. Dependency Injection
2. High Level Abstraction and Simplified Development
3. Enhanced Integration & Ecosystem
4. AOP (Aspect Oriented Programming)
5. Easy to test the application
6. Scalabilty and maintability
etc


fields, methods,
constructors,
state, behavior
simple object
confgurations
life-cycle
bean object


Java Class
= POJO class
= JavaBean class
Configuration Metadata
= XML file
= Java class
= Annotations
etc
bean objects
Spring Container
(IoC Container)
We can access the objects

=> What is Spring Container :-
-> Spring container is the "Heart" or "Core Component" of spring framework
-> It is same like :-
= JVM which is used to execute java programs
= Servlet Container or JSP Container which is used to execute Servlet and JSP pages
= EJB Container which provides runtime environment for enterprise beans
-> Responsibilites of Spring Container :-
1. Instantiate bean object
2. Configure bean objects
3. Initialize bean objects
4. Manage bean life-cycle
5. Destroy bean objects
6. Dependency Injection
7. Resource Management
8. AOP (Aspect Oriented Programming)
etc
-> Types of Spring Container :-
1. BeanFactory (old)
2. ApplicationContext (new)
-> Working of Spring Container :-

=> What do you mean by configurations in spring ?

-> Configurations refers to the settings or instructions provided to the spring framework (spring container) to define and manage various aspects of an application

-> There are a lot configurations in spring :-

1. Bean definations
2. Dependency injection
3. Bean life-cycle
4. Bean autowiring
5. Bean post-processing
6. Component scanning
7. Database configurations
8. AOP
9. View resolvers
10. Security configurations
11. Property files

etc

-> In spring, we can provide configurations by :-

1. XML files
2. Java classes
3. Annotations
4. Property files
5. Environment variables
6. Command-line arguments
7. Profiles

etc

=> What is POJO class :-

-> POJO stands for "Plain Old Java Object"

-> It is simple java class which follows some basic conventions for encapsulation, modularity and maintability

-> Syntax :-

```
public class Student
{
    String name;
    public int rollno;
    private int marks;

    //getter and setter methods
}
```

=> What is JavaBean class :-

-> JavaBean class is the class which encapsulates many objects/properties into single unit

-> JavaBean class is a special type of POJO class which follows the following conventions :-

1. Class must be public
2. It must inherit "Serializable" interface
3. It must contain public no-arugment constructor
4. All the properties must be private
5. It should have public getter and setter methods

-> Syntax :-

```
public class Student implements Serializable
{
    public Student(){}

    private String name;
    private int rollno;
    private int marks;

    //public getter and setter methods
}
```

-> NOTE : All JavaBean classes are POJO classes but all POJO classes are not JavaBean classes

=> What is difference between POJO class and JavaBean class :-

1. POJO class is the class whch does not have any restriction
   JavaBean class is the POJO class having some restrictions

2. POJO class may/may not inherit Serializable interface
   JavaBean class must inherit Serializable interface

3. POJO class may/may not contain no-arugment constructor
   JavaBean class must have public no-arugment constructor

4. In POJO class, fields or properties can have any visibility i.e. private, public, default
   In JavaBean class, fields or properties must be private

5. In POJO classes, fields or properties can be accessed by their names
   In JavaBean classes, fields or properties can be accessed only by getter and setter methods

6. POJO class does not have control on members
   JavaBean class have full control on members

7. We cannot use annotations in POJO classes
   We can use annotations in JavaBean classes

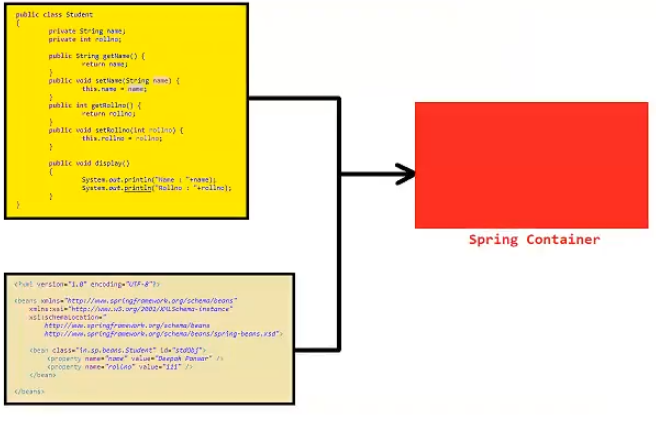
8. We cannot provide ay business logic in POJO classes
   We can provide business logic in JavaBean classes

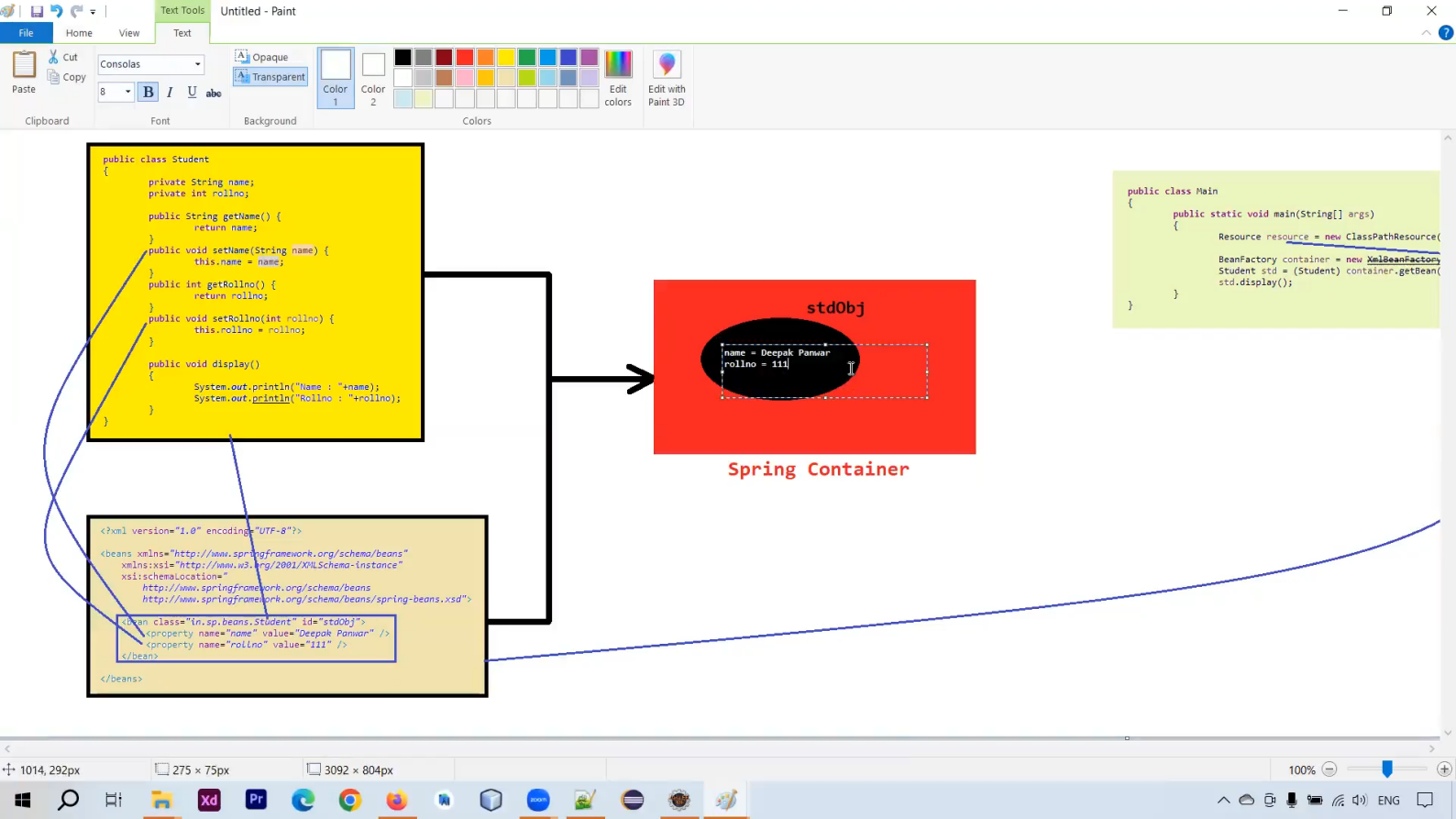
=> What is configuration metadata file :-
-> It is also known as spring configuration file which contains the configuration metadata for our spring application
-> It serves as a central repository to configure beans, dependencies, scopes and other application-specific settings
-> NOTE : If configuration metadata file is an XML file, then name should be "applicationContext.xml"

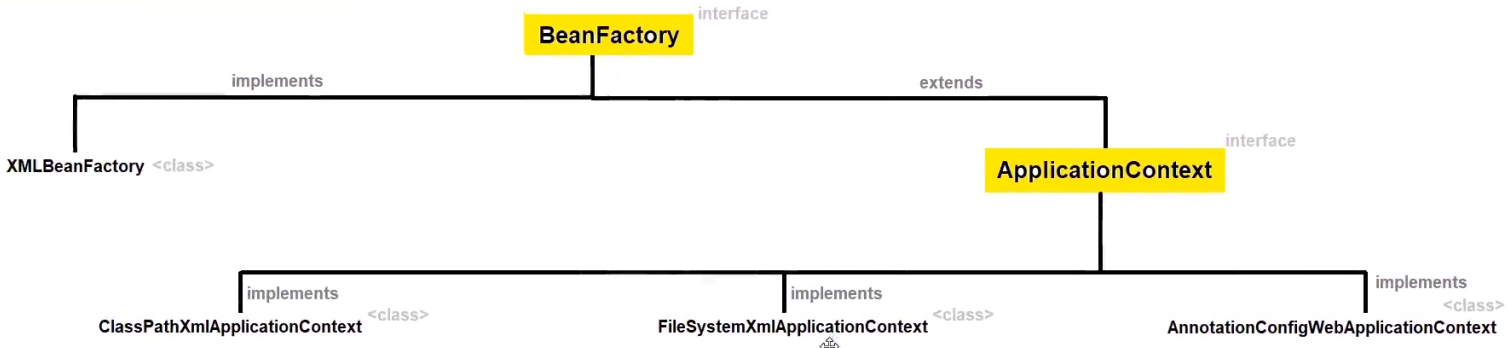
=> Package Name :-
    = Organisation Website : www.google.com
    = com.google.beans

    = www.smartprogramming.in --> in.sp.beans

=> What is Resource ?
    -> Whenever we need to load or read any external file/resource i.e. xml file, text file, properties file, image etc then we use Resource
    -> Resource is an pre-defined interface present in "org.springframework.core.io" package
    -> Spring framework has provided some implemented classes for Resource interface :-
        1. ClassPathResource
        2. URLResource
        3. InputStreamResource
        4. ByteArrayResource
        5. FileSystemResource
           etc


public class Student
{
private String name;
private int rollno;
public String getName() {
return name;
}
public void setName(String name) {
this.name = name;
}
public int getRollno() {
return rollno;
}
public void setRollno(int rollno) {
this.rollno = rollno;
}
public void display()
{
System.out.println("Name : "+name);
System.out.println("Rollno : "+rollno);
}
}
Spring Container
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
<bean class="in.sp.beans.Student" id="std001">
<property name="name" value="Deepak Panwar" />
<property name="rollno" value="111" />
</bean>
</beans>
public class Main
{
public static void main(String[] args)
{
Resource resource = new ClassPathResource("/in/sp/resources/applicationContext.xml");
BeanFactory container = new XmlBeanFactory(resource);
Student std = (Student) container.getBean("std001");
std.display();
}
}


Text Tools
Untitled - Paint
File
Home
View
Text
Cut
Copy
Paste
Consolas
8
Opaque
Transparent
Color 1
Color 2
Edit colors
Edit with Paint 3D
Clipboard
Font
Background
Colors
public class Student
{
private String name;
private int rollno;
public String getName() {
return name;
}
public void setName(String name) {
this.name = name;
}
public int getRollno() {
return rollno;
}
public void setRollno(int rollno) {
this.rollno = rollno;
}
public void display()
{
System.out.println("Name : "+name);
System.out.println("Rollno : "+rollno);
}
}
public class Main
{
public static void main(String[] args)
{
Resource resource = new ClassPathResource(
BeanFactory container = new XmlBeanFactory
Student std = (Student) container.getBean(
std.display();
}
}
stdObj
name = Deepak Panwar
rollno = 111
Spring Container
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
<bean class="in.sp.beans.Student" id="stdObj">
<property name="name" value="Deepak Panwar" />
<property name="rollno" value="111" />
</bean>
</beans>
1014, 292px
275 × 75px
3092 × 804px
100%
ENG


BeanFactory
interface
implements
extends
XMLBeanFactory <class>
ApplicationContext
interface
implements
ClassPathXmlApplicationContext <class>
implements
FileSystemXmlApplicationContext <class>
implements
AnnotationConfigWebApplicationContext <class>

=> What is BeanFactory ?
    -> It is the core interface in the Spring Framework for managing and accessing the beans
    -> It serves as a "Spring Container" that instantiate, configure, manage bean life cycle etc

=> What is ApplicationContext ?
    -> It is the sub-interface of BeanFactory for managing and accessing bean objects
    -> It serves as a "Spring Container" which provides more functionailites as compared to BeanFactory
    -> In simple terms we can say that it is an advanced spring container as compared to BeanFactory

=> Hierarchy of Spring Container :-
    ->

=> Difference between BeanFactory & ApplicationContext ?

1. BeanFactory is the core container or fundamental container
   ApplicationContext is an advanced spring container which provides all the functionailites of BeanFactory container

2. BeanFactory creates the bean object when we call getBean(-) method and thus it is known as lazy instantiation
   ApplicationContext creates the bean object when the container gets started and thus it is known as eager instantiation

3. BeanFactory supports only singleton and prototype scope
   ApplicationContext supports singleton, prototype, request, session scopes

4. BeanFactory does not support I18N functionality
   ApplicationContext supports I18N functionality

5. BeanFactory does not support AOP and ORM
   ApplicationContext supports AOP and ORM

6. BeanFactory does not support annotations
   ApplicationContext supports annotations

7. BeanFactory is suitable for Standalone Applications
   ApplicationContext is suitable for Enterprise Applications

==========

```
<!--
<bean class="in.sp.beans.Student" name="stdObj1, stdObj2">
    <property name="name" value="Deepak Panwar" />
    <property name="rollno" value="222" />
</bean>
-->

<!--
<bean class="in.sp.beans.Student" name="stdObj1 stdObj2">
    <property name="name" value="Deepak Panwar" />
    <property name="rollno" value="222" />
</bean>
-->

<bean class="in.sp.beans.Student" name="stdObj1; stdObj2">
    <property name="name" value="Deepak Panwar" />
    <property name="rollno" value="222" />
</bean>
```

=> What is bean ?

-> Bean are the objects that form the backbone of our spring application which is managed by Spring Container

-> Beans are created with the configuration details/metadata that we provides to spring container using spring configuration file i.e. .xml file or .java file

-> There are some important attributes related to bean objects :-

1. Class
2. Id or Name
3. Property Values
4. Constructor Arguments
5. Scope
6. Initalization and Destruction Callbacks
7. Lazy Initialization
8. Bean Post-Processors
8. Autowiring
9. Profiles

   etc

```
<!--
<bean class="in.sp.beans.Student" name="stdObj1; stdObj2">
    <property name="name" value="Deepak Panwar" />
    <property name="rollno" value="222" />
</bean>
-->

<!--
<bean class="in.sp.beans.Student" name="stdObj1" id="stdObj1">
    <property name="name" value="Deepak Panwar" />
    <property name="rollno" value="222" />
</bean>
-->
```

```
<bean class="in.sp.beans.Student" name="stdObj1">
    <property name="name" value="Deepak Panwar" />
    <property name="rollno" value="222" />
</bean>

<bean class="in.sp.beans.Student" name="stdObj1">
    <property name="name" value="Amit Sharma" />
    <property name="rollno" value="333" />
</bean>
```

=> What is id and name attributes for bean object ?

-> id :-

= It specifies the unique identity of bean object

-> name :-

= It specifies the unique identity of bean object but it is more flexible as compared to id attribute

= Flexibilities that are provided by name attribute are :-

1. We can provide multiple names for one bean object
2. We can separate the multiple bean names by comma(,) or semi-colon(;) or space
3. We can provide same bean object name in name and id attribute
4. We can provide same name to one bean object but same name cannot be provided to multiple bean objects

applicationContext.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean class="in.sp.beans.Student" id="stdObj1">
      <property name="name" value="Deepak Panwar" />
      <property name="rollno" value="222" />
    </bean>

    <bean class="in.sp.beans.Student" id="stdObj2">
      <property name="name" value="Amit Sharma" />
      <property name="rollno" value="333" />
    </bean>

</beans>
```

SpringConfigFile.java

```java
@Configuration
public class SpringConfigFile
{
      @Bean(name = "stdObj1")
      public Student createBeanObj1()
      {
            Student std = new Student();

            std.setName("Deepesh Panwar");
            std.setRollno(444);

            return std;
      }

      @Bean(name = "stdObj2")
      public Student createBeanObj2()
      {
            Student std = new Student();

            std.setName("Kamal Sharma");
            std.setRollno(555);

            return std;
      }
}
```

=> Java Based Configuration :-
-> Before Spring 3.0 version, it was compulsory to provide spring configurations metadata by using xml file. But from spring 3.0 version, its not compulsory to create xml file. We can also provide configurations metadata by using java class
-> How to achieve java based configurations :-
1. Create java class i.e. configuration class and mark it as @Configuration annotation
2. Create one or more methods (which returns bean object) and mark it as @Bean annotation
3. Create object of AnnotationConfigApplicationContext class and access the bean object

-> NOTE : If we dont provide bean object name manually then bean name will be same as that of method name. If we want to declare bean name manually then we can use @Bean(name = "beanObjectName")

=> What is @Configuration annotation ?
-> @Configuration annotation is used with class
-> When spring container starts, it will check all the java classes marked with @Configuration. Then it will load the class into memory and process them to create bean defination/configurations

=> What is @Bean annotation ?
-> @Bean annotation is used with methods
-> @Bean methods are responsible to create and configure bean objects.
-> When spring container starts, it will invokes each @Bean method and create the bean objects
-> By default bean object name is same as method name but if we want to change the bean object name then we can use name attribute i.e. @Bean(name = "beanObjectName")

```
5
6 @Component("stdObj")
7 public class Student
8 {
```

=> @Component :-
    -> It is also known as "stereotype annotation"
    -> It is used to mark the class as a spring-managed component. The spring container is responsible for creating, configuring and managing the components inluding their life-cycle, dependency injection etc
    -> By default @Component scope is "singleton scope"

=> Some examples of spring-managed components are :-
    1. @Configuration
    2. @Bean
    3. @Component
        = @Controller
        = @Service
        = @Repository
    4. @Autowired
    5. @Aspect
       etc

=> Different ways to create bean objects and property configuration :-

1. XML file

```
<bean class="fully qualified JavaBean class name" id="beanId">
    <property name="property_name" value="property_value" />
    <property name="property_name" value="property_value" />
</bean>
```

2. Java class :-

```
@Configuration
class JavaConfigFile
{
    public JavaBean m1()
    {
        JavaBean obj = new JavaBean();

        obj.setXXX(-);
        obj.setXXX(-);

        return obj;
    }
}
```

3. Annotations :-

```
public class JavaBean
```

3. Annotations :-

```
@Component
public class JavaBean
{
    @Value("--")
    private String property_name;
    --
}
```

NOTE : we have to either register the JavaBean class or scan the packages

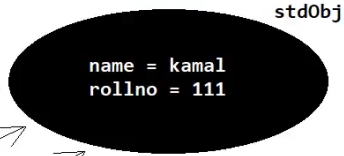
=> Bean Scope :-
    -> Bean Scope defines the visibility or accessability of that bean in the context we use it
    -> We can provide bean scope by using "scope attribute" or "@Scope annotation"
    -> There are total 7 scopes :-
        1. "singleton" scope
        2. "prototype" scope

        3. "request" scope
        4. "session" scope
        5. "globalSession" scope
        6. "application" scope
        7. "webSocket" scope

    -> NOTE : By default, beans are singleton scope

=> "singleton" scope :-
    -> It is the default scope of bean object
    -> In this scope only one instance will be created for a single bean defination and that same object will be shared for each request made for that bean using getBean(-) method
    -> Program

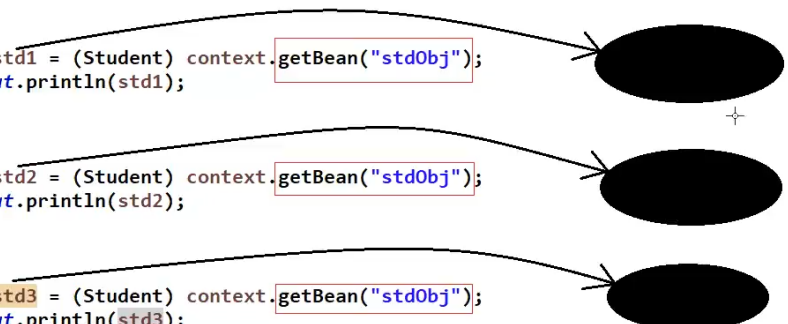
```xml
<bean class="in.sp.beans.Student" id="stdObj">
    <property name="name" value="Kamal" />
    <property name="rollno" value="111" />
</bean>
```




```java
ApplicationContext context = new ClassPathXmlApplicationContext("/in/sp/resources/applicationContext.xml");


Student std1 = (Student) context.getBean("stdObj");
System.out.println(std1);


Student std2 = (Student) context.getBean("stdObj");
System.out.println(std2);


Student std3 = (Student) context.getBean("stdObj");
System.out.println(std3);
```

```xml
<bean class="in.sp.beans.Student" id="stdObj" scope="prototype">
    <property name="name" value="Kamal" />
    <property name="rollno" value="111" />
</bean>
```

```
ApplicationContext context = new ClassPathXmlApplicationContext("/in/sp/resources/applicationContext.xml");


Student std1 = (Student) context.getBean("stdObj");
System.out.println(std1);


Student std2 = (Student) context.getBean("stdObj");
System.out.println(std2);


Student std3 = (Student) context.getBean("stdObj");
System.out.println(std3);
```



=> "prototype" scope :-
    -> In this scope a new instance is created for a single bean defination and the new object will be shared for each request made for that bean using getBean(-) method
    -> Program

=> @Component :-
    -> It is also known as "stereotype annotation"
    -> It is used to mark the class as a spring-managed component. The spring container is responsible for creating, configuring and managing the components inluding their life-cycle, dependency injection etc
    -> By default @Component scope is "singleton scope"

=> Some examples of spring-managed components are :-
    1. @Configuration
    2. @Bean
    3. @Component
        = @Controller
        = @Service
        = @Repository
    4. @Autowired
    5. @Aspect
       etc

=> Different ways to create bean objects and property configuration :-

1. XML file

```
<bean class="fully qualified JavaBean class name" id="beanId">
    <property name="property_name" value="property_value" />
    <property name="property_name" value="property_value" />
</bean>
```

2. Java class :-

```
@Configuration
class JavaConfigFile
{
    public JavaBean m1()
    {
        JavaBean obj = new JavaBean();

        obj.setXXX(-);
        obj.setXXX(-);

        return obj;
    }
}
```

3. Annotations :-

```
public class JavaBean
```

3. Annotations :-

```
@Component
public class JavaBean
{
    @Value("--")
    private String property_name;
    --
}
```

NOTE : we have to either register the JavaBean class or scan the packages

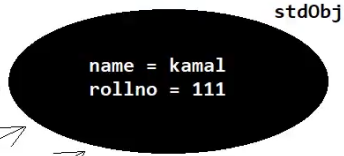
=> Bean Scope :-
    -> Bean Scope defines the visibility or accessability of that bean in the context we use it
    -> We can provide bean scope by using "scope attribute" or "@Scope annotation"
    -> There are total 7 scopes :-
        1. "singleton" scope
        2. "prototype" scope

        3. "request" scope
        4. "session" scope
        5. "globalSession" scope
        6. "application" scope
        7. "webSocket" scope

    -> NOTE : By default, beans are singleton scope

=> "singleton" scope :-
    -> It is the default scope of bean object
    -> In this scope only one instance will be created for a single bean defination and that same object will be shared for each request made for that bean using getBean(-) method
    -> Program

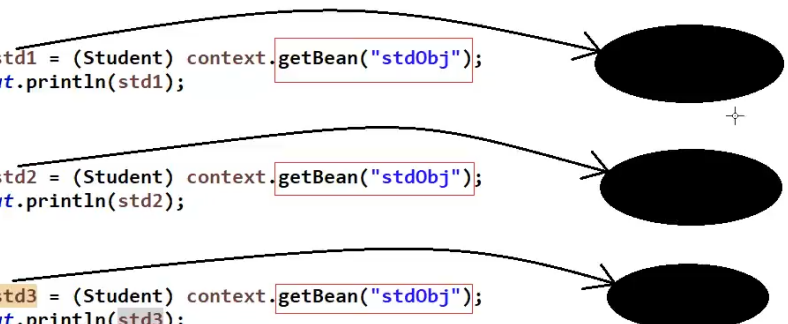
```xml
<bean class="in.sp.beans.Student" id="stdObj">
    <property name="name" value="Kamal" />
    <property name="rollno" value="111" />
</bean>
```




```java
ApplicationContext context = new ClassPathXmlApplicationContext("/in/sp/resources/applicationContext.xml");


Student std1 = (Student) context.getBean("stdObj");
System.out.println(std1);


Student std2 = (Student) context.getBean("stdObj");
System.out.println(std2);


Student std3 = (Student) context.getBean("stdObj");
System.out.println(std3);
```

```xml
<bean class="in.sp.beans.Student" id="stdObj" scope="prototype">
    <property name="name" value="Kamal" />
    <property name="rollno" value="111" />
</bean>
```

```
ApplicationContext context = new ClassPathXmlApplicationContext("/in/sp/resources/applicationContext.xml");


Student std1 = (Student) context.getBean("stdObj");
System.out.println(std1);


Student std2 = (Student) context.getBean("stdObj");
System.out.println(std2);


Student std3 = (Student) context.getBean("stdObj");
System.out.println(std3);
```



=> "prototype" scope :-
    -> In this scope a new instance is created for a single bean defination and the new object will be shared for each request made for that bean using getBean(-) method
    -> Program

=> Bean Life Cycle :-
    1. Loading Bean Definations
    2. Bean Instantiation
    3. Bean Initialization
    4. Bean Usage
    5. Bean Destruction

---

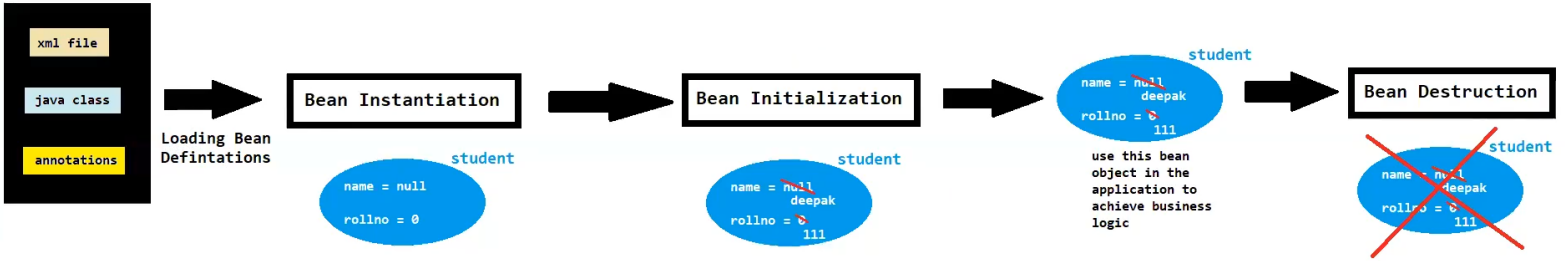
1. Loading Bean Definations :-
    -> Bean definations are the configurations (blueprint or settings) that defines how bean object should be created. It includes the information about the class to instantiate, property configurations, dependency injection and other configurations
    -> Bean definations can be provided by xml file or java class or annotations
    -> It is the process of reading and parsing the configuration files to create bean definations for the beans that will be managed by the spring container

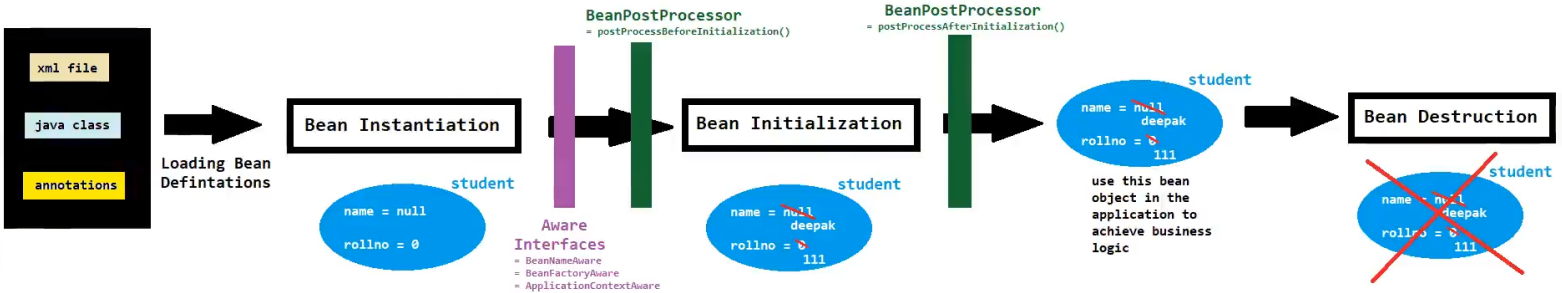
2. Bean Instantiation :-
   -> In this phase, spring container will create an instance of the bean based on it bean definations
   -> How bean objects are created ?
      a. using default constructor or no-argument constructor
      b. using static factory method
      c. using instance factory method
   -> In this phase, bean objects are initialized with default values based on the data types of the properties in the JavaBean class
   -> In this phase, the container also injects the required dependencies into the bean object by any following way :-
      a. Setter method DI
      b. Constructor DI

3. Bean Initialization :-
   -> In this phase bean object is initialized by its original values
   -> How bean objects are initialized ?
      a. using property tags
      b. using explicit ways
         i. using custom init() method
         ii. using afterPropertiesSet() method of InitializingBean callback interface
         iii. using @PostConstruct annotation

3. Bean Initialization :-
   -> In this phase bean object is initialized by its original values
   -> How bean objects are initialized ?
      a. using property tags
      b. using explicit ways
         i. using custom init() method
         ii. using afterPropertiesSet() method of InitializingBean callback interface
         iii. using @PostConstruct annotation (jar file is needed javax.annotation-api-xxx.jar)

4. Bean Usage :-
   -> Once the bean is fully initialized, it is ready to be used in our application.
   -> Beans can be retrieved from the spring container and can be used for business logic in our application


xml file
java class
annotations
Loading Bean Defintations
Bean Instantiation
student
name = null
rollno = 0
Bean Initialization
student
name = null
deepak
rollno = 0
111
student
name = null
deepak
rollno = 0
111
use this bean object in the application to achieve business logic
Bean Destruction
student
name = null
deepak
rollno = 0
111


xml file
java class
annotations
Loading Bean Defintations
Bean Instantiation
student
name = null
rollno = 0
BeanPostProcessor
= postProcessBeforeInitialization()
Aware Interfaces
= BeanNameAware
= BeanFactoryAware
= ApplicationContextAware
Bean Initialization
student
name = null
deepak
rollno = 0
111
BeanPostProcessor
= postProcessAfterInitialization()
student
name = null
deepak
rollno = 0
111
use this bean object in the application to achieve business logic
Bean Destruction
student
name = null
deepak
rollno = 0
111

Student.java | applicationContext.xml | student.properties

```
1 student.name=Deepak
2 student.rollno=101
3 student.subjmarks.C=87
4 student.subjmarks.Cpp=83
5 student.subjmarks.Java=97
6 subject.subjmarks.Python=91
```

```
6
7    <bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
8        <property name="location" value="/in/sp/resources/student.properties" />
9    </bean>
10
11   <bean class="in.sp.beans.Student" id="stdId">
12       <property name="name" value="${student.name}" />
13       <property name="rollno" value="${student.rollno}" />
14       <property name="subjmarks">
15           <map>
16               <entry key="C" value="${student.subjmarks.C}" />
17               <entry key="C++" value="${student.subjmarks.Cpp}" />
18               <entry key="Java" value="${student.subjmarks.Java}" />
19               <entry key="Python" value="${student.subjmarks.Python}" />
20           </map>
21       </property>
```

```
@PropertySource("/in/sp/resources/student.properties")
```

```
@Value("#{${unknownMap : {'C': 1, 'Java': 2}}}")
private Map<String, Integer> subjmarks;
```

=> Property Configurations :-
    -> Property Configurations is the process by which we set the values in the bean properties
    -> We can set the property values either from xml file or java or properties file

=> Spring Configurations :-
    -> It is the process of providing the spring application configurations that how our spring application will work

=> What is properties file ?
    -> A property file is a simple text file commonly used to store configurations data in key-value pair
    -> Advantage of properties file :-
        1. Easy to update or modify the configurations
        2. We dont need to restart the application if we change or modify the configurations
        3. Easy to test the application
           etc

==========

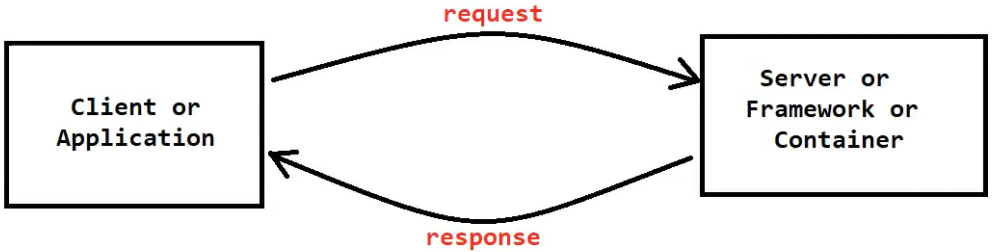
```
<bean class="in.sp.beans.Student" id="stdId">
    <constructor-arg value="Deepak" />
    <constructor-arg value="111" />
    <constructor-arg value="94.8" />
</bean>
```

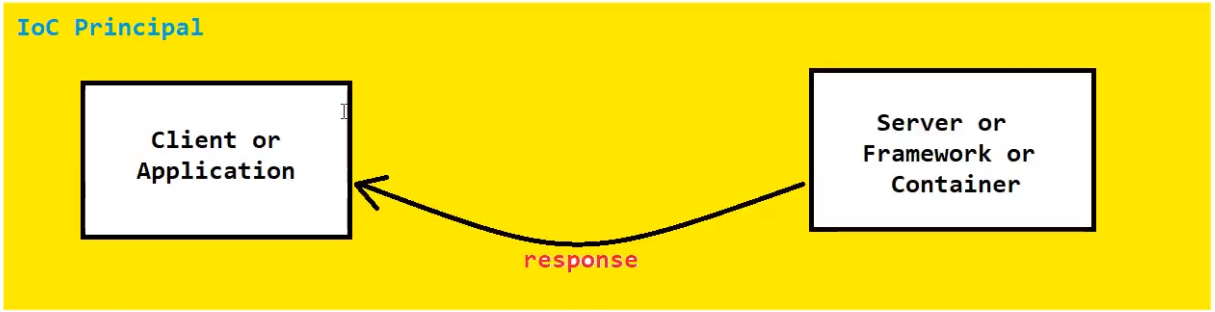
```
<bean class="in.sp.beans.Student" id="stdId">
    <constructor-arg value="111" index="1" />
    <constructor-arg value="Deepak" />
    <constructor-arg value="94.8" />
</bean>
```

```
<bean class="in.sp.beans.Student" id="stdId">
    <constructor-arg value="111" type="int" />
    <constructor-arg value="94.8" type="float" />
    <constructor-arg value="Deepak" />
</bean>
```

=> Property Configurations :-
    -> Property Configurations is the process by which we set the values in the bean properties
    -> We can set the property values either from xml file or java or properties file

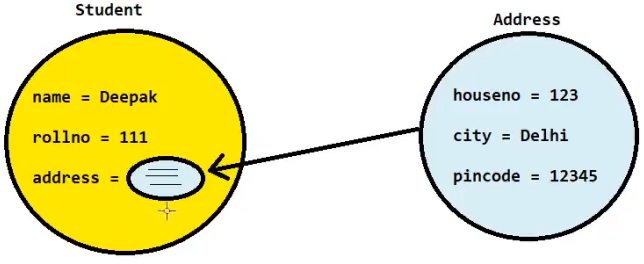
    -> We can provide the property configuration in xml file by 2 ways :-
        1. By using <property> tag
        2. By using <constructor-arg> tag


request
Client or Application
Server or Framework or Container
response


IoC Principal
Client or Application
Server or Framework or Container
response

=> Inversion Of Control (IoC) :-
-> Inversion of Control is a design pattern/principal that focus on inverting the conrol flow of an application
-> It shifts the responsibility of managing the flow of execution and the lifecycle of objects from application itself to external entity i.e. framework or container
-> It identifies the client required dependencies or services and then it will create and inject the required dependencies or service to the application without client request
-> Spring Container works on the basis of IoC principal and thus it is also known as IoC Container
-> Advantages of IoC principal :-
    1. Classes are loosly coupled
    2. Modularity can be achieved
    3. Easier to test and maintain the application
       etc
-> In spring IoC principal can be achieved by following :-
    1. Dependency Injection (DI)
    2. Service Locator
    3. Contextualized Lookup
    4. Template Method Design Pattern
    5. Event Based IoC
       etc
-> NOTE : From above, only DI is most commonly IoC principal used in spring

```
class Student
{
    private String name;
    priavte int rollno;
    private Address address;

    //getter and setter methods

    //other methods
}
```




```
class Address
{
    private int houseno;
    private String city;
    private int pincode;

    //getter and setter methods

    //other methods
}
```

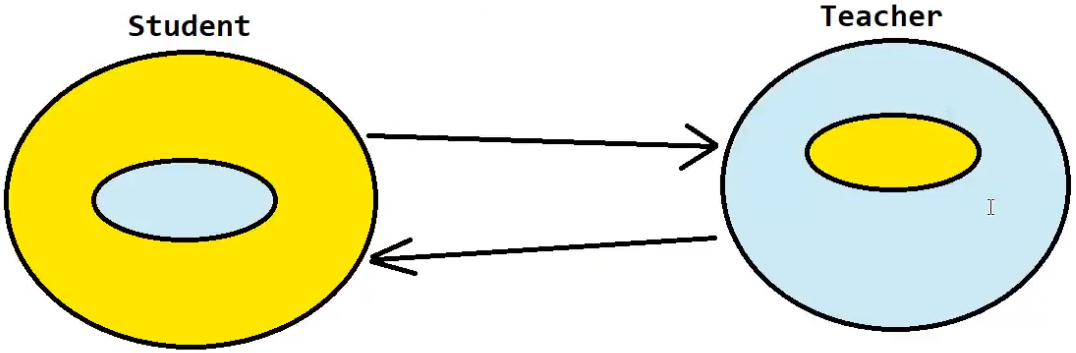
```
<!-- bean definitions here -->
 <bean class ="me.beans.Address" id="addId">
    <property name = "addName" value = "Shukhi Bhawan"/>
    <property name = "houseNo" value = "012"/>
    <property name = "city" value="Dhanbad"/>
</bean>
<bean class ="me.beans.Student" id="stdId">
    <property name = "name" value = "Piyush"/>
    <property name = "id" value = "95"/>
    <property name = "address" ref="addId"/>
</bean>
```

=> Dependency Injection (DI) :-
    -> Dependency Injection is a design pattern that is used to implement IoC principal
    -> Dependency Injection main functionality is to "inject" one object into another object
    -> How to achieve DI in Spring Configuration File (xml) :-
        = We can achieve DI by 2 ways :-
            1. Setter Method DI
            2. Constructor DI

==========

=> What is difference between Setter Method DI and Constructor DI :-
    1. How dependency is injected :-
        = Setter Method DI uses setter methods i.e. setXXX() method to inject the dependency
        = Constructor DI uses constructor to inject the dependency
    2. Readability :-
        = Setter Method DI has more readability because we have to provide the property name and its value
        = Constructor DI has less readability because we dont provide the property name with value
    3. Partial Dependency :-
        = Partial Dependency is possible in case of Setter Method DI
        = Partial Dependency is not possible in case of Constructor DI
    4. Circular DI :-
        = We can achieve Circular DI using Setter Method DI
        = We cannot achieve Circular DI using Constructor DI

Circular DI



```
17
18 !--this is the alternative and basically easier way to set the data bus upar mein ek
19 chema paste krna hai aur kuch nhi  -->
20
21 !--<bean class = "me.beans.Student" id = "std"
22     p:name="piyush"
23     p:rollno="233"
24     p:marks="92.3"
25 />-->
26
```

```
<bean class = "me.beans.Address" id = "add"
    p:addName="Sukhi Bhavan"
    p:houseNo="233"
    p:city="Dhanbad"
    />

<!--if there is any class related to any another class then we have to use ref as shown below-
<bean class = "me.beans.Student" id = "std"
    p:name="piyush"
    p:rollno="233"
    p:marks="92.3"
    p:address-ref="add"
    />
```

=> p-namespace :-
-> When we have to inject dependencies in bean object using setter method then we have to provide <property> tag. More dependencies more <property> tag and due to this the code become lengthy which is not good.
-> To solve this problem, spring has provided one feature or shortcut i.e. p-namespace
-> How to use p-namespace :-
1. Provide p-namespace declaration in spring configuration file
xmlns:p="http://www.springframework.org/schema/p"
2. Then we can provide the dependencies in bean tag by p:property_name="value" OR p:property_name-ref="value"

=> c-namespace :-
-> When we have to inject dependencies in bean object using constructor then we have to provide <constructor-arg> tag. More dependencies more <constructor-arg> tags and due to this the code becomes lengthy which is not good
-> To solve this problem, spring has provided one feature or shortcut i.e. c-namespace
-> How to use c-namespace :-
1. Provide c-namespace declaration in spring configuration file
2. Then we can provide the dependencies in bean tag by c:property_name="value" OR c:_indexPosition="value" OR c:property_name-ref="value"

```
==========================================================================================

=> Depedency Injection using Java Configuration File :-
    -> How to achieve DI in Spring Configuration File (java) :-
        = We can achieve DI by 2 ways :-
            1. Setter Method DI
            2. Constructor DI
    -> Programs
```

=> Dependency Injection (DI) :-
    -> DI is the process by which we can inject one bean object into another bean object
    -> DI can be achieved by 2 ways :-
        1. Setter Method DI
        2. Constructor DI
        (NOTE : both these have been done by using xml and java based configuration)

    -> Till now we have achieved DI by explicit ways

=> Autowiring :-
    -> Autowiring is the feature of Spring Framework by which we can achieve DI automatically
    -> Advantage :-
        = It requires less code
    -> Disadvantage :-
        = There is no control of programmer
        = It can be achieved only on non-primitive or user-defined data types (excluding String), not on primitive data types
    -> How can we achieve autowiring :-
        = We can achieve autowiring by 4 ways :-
            1. XML Based Autowiring
            2. Annotation Based Autowiring
            3. Java Based Autowiring
            4. Component Scanning

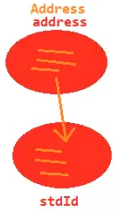
```java
public class Student
{
	private String name;
	private int rollno;
	private Address address;

	//getter and setter methods

	public void display()
	{
		System.out.println("Name : "+name);
		System.out.println("Rollno : "+rollno);
		System.out.println("Address : "+address);
	}
}
```

```java
public class Address
{
	private int houseno;
	private String city;
	private int pincode;

	//getter and setter methods

	@Override
	public String toString()
	{
		return "#"+houseno+", "+city+" - "+pincode;
	}
}
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans ------------>

  <bean class="in.sp.beans.Address" id="address">
    <property name="houseno" value="123" />
    <property name="city" value="Delhi" />
    <property name="pincode" value="12345" />
  </bean>

  <bean class="in.sp.beans.Student" id="stdId" autowire="byType">
    <property name="name" value="Deepak" />
    <property name="rollno" value="111" />
  </bean>

</beans>
```

```
<bean class = "me.beans.Address" id = "address">
    <property name = "addName" value = "Sukhi Bhawan"/>
    <property name = "houseNo" value = "113"/>
    <property name = "city" value = "Dhanbad"/>
</bean>

    <!--yaha pr hum agar bytype use kr rahe hai to jo Address and jo uska
    bean class provide kr rahe hai to hi autowiring achieve ho payega
       achive ho payega-->
```

```java
public class Student
{
	private String name;
	private int rollno;
	private Address address;

	//getter and setter methods

	public void display()
	{
		System.out.println("Name : "+name);
		System.out.println("Rollno : "+rollno);
		System.out.println("Address : "+address);
	}
}
```




```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans ------------>

    <bean class="in.sp.beans.Address" id="address">
      <property name="houseno" value="123" />
      <property name="city" value="Delhi" />
      <property name="pincode" value="12345" />
    </bean>

    <bean class="in.sp.beans.Student" id="stdId" autowire="byType">
      <property name="name" value="Deepak" />
      <property name="rollno" value="111" />
    </bean>

</beans>
```

```java
public class Address
{
	private int houseno;
	private String city;
	private int pincode;

	//getter and setter methods

	@Override
	public String toString()
	{
		return "#"+houseno+", "+city+" - "+pincode;
	}
}
```

```java
public class Main
{
	public static void main(String[] args)
	{
		ApplicationContext context = new ClassPathXmlApplic----("/in/sp/resources/applicationContext.xml");

		Student std = (Student) context.getBean("stdId");
		std.display();
	}
}
```

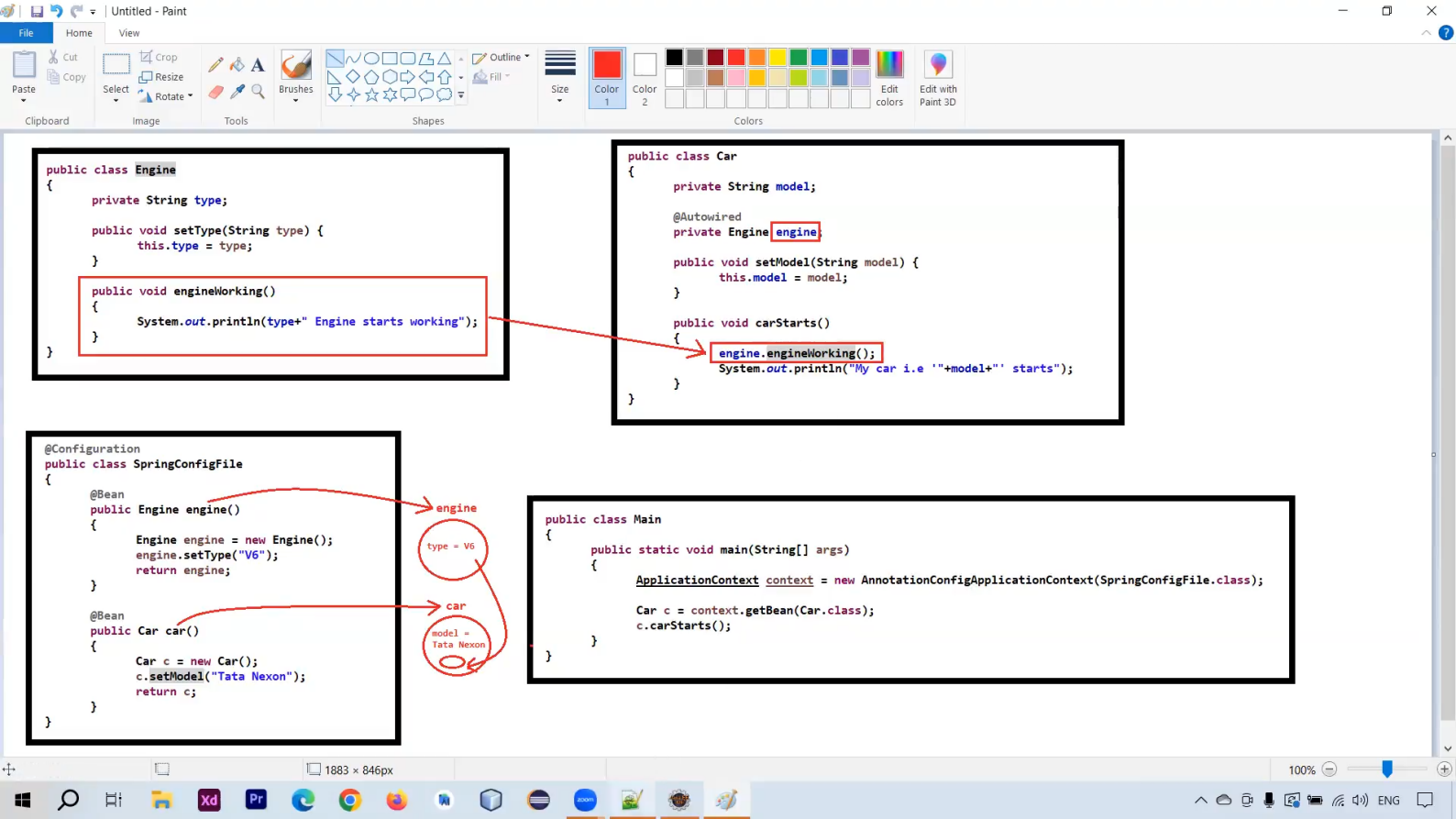
```
    <bean class="in.sp.beans.Address" id="addrId1">
        <property name="houseno" value="123" />
        <property name="city" value="Delhi" />
        <property name="pincode" value="12345" />
    </bean>

    <bean class="in.sp.beans.Address" id="addrId2" autowire-candidate="false">
        <property name="houseno" value="678" />
        <property name="city" value="Mumbai" />
        <property name="pincode" value="67890" />
    </bean>

    <bean class="in.sp.beans.Student" id="stdId" autowire="byType">
        <property name="name" value="Deepak" />
        <property name="rollno" value="111" />
    </bean>
```

```
<bean class="in.sp.beans.Address" id="address">
    <constructor-arg value="123" />
    <constructor-arg value="Delhi" />
    <constructor-arg value="12345" />
</bean>

<bean class="in.sp.beans.Student" id="stdId" autowire="constructor">
    <constructor-arg value="Deepak" index="0" />
    <constructor-arg value="111" index="1" />
</bean>
```

=> XML Based Autowiring :-
-> In case of XML Based Autowiring, we dont need to use "ref" attribute in <property> or <constructor-arg> tag

-> We can achieve XML based autowiring by using "autowire" attribute in <bean> tag i.e.
<bean class="----" id="----" autowire="--modes--">
-> Modes of autowire attribute :-
1. no :-
= It is default autowiring mode
= It simply means that we dont want to achieve autowiring
2. byName
= In this case we will achieve autowiring by matching "property name" of bean object and "bean id" in spring configuration file
= It uses "Setter Method DI" internally
3. byType
= In this case we will achieve autowiring by matching the data-types i.e. "data-types" in bean class should be same as that of "class" in <bean> tag
= It uses "Setter Method DI" internally
= In this case, if we have create multiple bean objects of one class, then which class it will inject, confusion will occur. To remove this confusion we can we one attribute i.e. "autowire-candidate" i.e. autowire-candidate="false". Whenever we will use this attribute with the bean, it will not participate in autowiring
4. constructor
= This is same as that of byType
= It internally use "Constructor DI"
5. autodeduct
= It is depricated from spring 3.x version

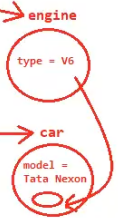
```
@Autowired
@Qualifier("createAddrObj1")
```


Untitled - Paint
File
Home
View
Cut
Copy
Paste
Crop
Resize
Rotate
Select
Brushes
Outline
Fill
Size
Color 1
Color 2
Edit colors
Edit with Paint 3D
Clipboard
Image
Tools
Shapes
Colors
public class Engine
{
private String type;
public void setType(String type) {
this.type = type;
}
public void engineWorking()
{
System.out.println(type+" Engine starts working");
}
}
public class Car
{
private String model;
@Autowired
private Engine engine;
public void setModel(String model) {
this.model = model;
}
public void carStarts()
{
engine.engineWorking();
System.out.println("My car i.e '"+model+"' starts");
}
}
@Configuration
public class SpringConfigFile
{
@Bean
public Engine engine()
{
Engine engine = new Engine();
engine.setType("V6");
return engine;
}
@Bean
public Car car()
{
Car c = new Car();
c.setModel("Tata Nexon");
return c;
}
}
engine
type = V6
car
model = Tata Nexon
public class Main
{
public static void main(String[] args)
{
ApplicationContext context = new AnnotationConfigApplicationContext(SpringConfigFile.class);
Car c = context.getBean(Car.class);
c.carStarts();
}
}
1883 × 846px
100%
ENG

```java
public class Engine
{
	private String type;

	public void setType(String type) {
		this.type = type;
	}

	public void engineWorking()
	{
		System.out.println(type+" Engine starts working");
	}
}
```

```java
public class Car
{
	private String model;

	@Autowired
	private Engine engine;

	public void setModel(String model) {
		this.model = model;
	}

	public void carStarts()
	{
		engine.engineWorking();
		System.out.println("My car i.e '"+model+"' starts");
	}
}
```

```java
@Configuration
public class SpringConfigFile
{
	@Bean
	public Engine engine()
	{
		Engine engine = new Engine();
		engine.setType("V6");
		return engine;
	}

	@Bean
	public Car car()
	{
		Car c = new Car();
		c.setModel("Tata Nexon");
		return c;
	}
}
```




```java
public class Main
{
	public static void main(String[] args)
	{
		ApplicationContext context = new AnnotationConfigApplicationContext(SpringConfigFile.class);

		Car c = context.getBean(Car.class);
		c.carStarts();
	}
}
```

```
=> Java + Annotation Based Autowiring :-
    -> In this case we have to use @Autowired annotation
    -> @Autowired annotation can be used with field (property), setter method or constructor

    -> If there are 2 bean objects which are ready to be injected in the bean then there will be
    confusion. To remove this confusion we can use one annotation i.e. @Qualifier

    -> Note : We can use @Autowired annotation in case of XML spring configuration file
```

[

Field error in object 'target' on field 'name': rejected value [null];
codes [key_name.target.name,key_name.name,key_name.java.lang.String,key_name];
arguments [];
default message [Name is not valid],

Field error in object 'target' on field 'rollno': rejected value [0];
codes [key_rollno.target.rollno,key_rollno.rollno,key_rollno.int,key_rollno];
arguments [];
default message [Rollno cannot be 0],

Field error in object 'target' on field 'phoneno': rejected value [628383];
codes [key_phoneno.target.phoneno,key_phoneno.phoneno,key_phoneno.java.lang.String,key_phoneno];
arguments [];
default message [Phone no is not valid]

]

```
    List<ObjectError> list = dataBinder.getBindingResult().getAllErrors();

    if(list.isEmpty())
    {
        std.display();
    }
    else
    {
        for(ObjectError oe : list)
        {
            System.err.println(oe.getDefaultMessage());
        }
    }
}
```

```
=> Bean Validations :-
    -> Bean validation is the way by which we can check the proper data in the bean object
    -> We can achieve bean validations by multiple ways which are as follows :-
        1. Using "Validator" interface
        2. Using JSR-303 Bean Validation
        3. Using "@Valid" and "@Validate" annotations
        4. Using annotations of SpEL (Spring Expression Language)
           etc
```

==========

```
=> "Validator" interface :-
    -> Syntax :-
        interface Validator
        {
            public boolean supports(Class clazz)

            public void validate(Object obj, Errors errors)
        }
```

==========

```
=> DataBinder :-
    -> It is a class in sprnig used for data binding
    -> It is responsible for binding the data from HTTP request parameters to java objects

    -> NOTE :
        = It is not responsible for validation itself, but it can be configured with validators to
        perform validation on the bound data
        = The validation results are stored in a separate object of the type "BindingResult" which can be
        accessed after the data bind process
```

```
=> BindingResult :-
    -> It is an interface that represents the result of data binding and validations
    -> Implemented class of BindingResult interface are :-
        1. MapBindingResult
        2. BeanPropertyBindingResult
```

```
<bean class="in.sp.validators.StudentValidator" id="stdValId">
    <property name="resource" value="/in/sp/resources/error_messages.properties" />
</bean>
```

=> Logging :-

-> Logging is the process of tracking or recording important information, events, messages or issues that occur during the execution of our application.

-> The log files generated during logging process will help the developers or system administrators to monitor the application behaviour, diagous issues and track the errors

-> Use of logging :-

1. Error Tracking and Debugging :-
   = In web applications we can track the errors like error generated during form submission
2. Security Monitering :-
   = We can track security-related events such as failed to login attempts or unautorized acces attempts
3. Auditing & Compliance :-
   = A financial application might log all the financial transactions including the details of transaction, its time, location etc
4. Performace Analysis :-
   = We can track the time taken by our application to perform any event or respond
5. System Health Monitoring :-
   = In server enviornment we can track the memory usage, CPU load and other metrics
6. Deployment & Release Management :-
   = We can track the version number, time of releasing the application etc

etc

-> Where we can use logging :-

1. Software Development
   = Web Development
   = Mobile App Development
2. DevOps & Infrastructure :-
   = Server Applications
   = Databases
3. Networking & Security :-
   = Firewall & Security Appliances
   = Network Servers
4. Cloud Computing :-
   = Cloud Servers
   = Serverless Computing
5. Industrial Automation & IoT :-
   = Industrial Control Systems
   = IoT Devices

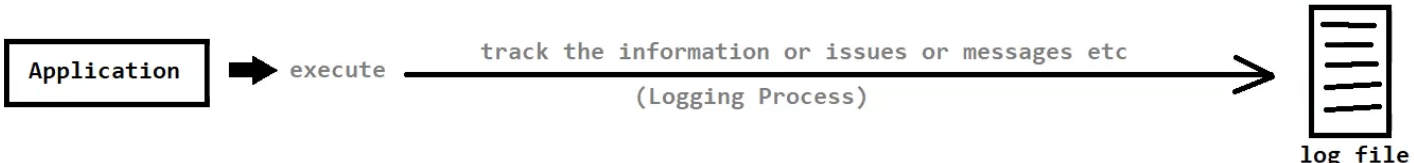
etc

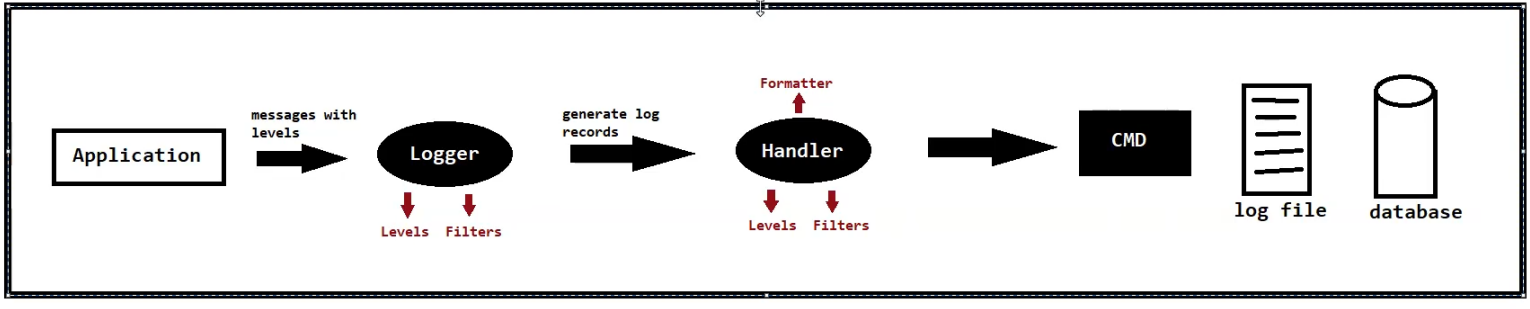
-> Logging process is supported by many languages i.e. java, python, php, JavaScript, Node JS etc

-> Logging was introduced in JDK 1.4 version
-> There are a lot of API's & Frameworks for logging :-

1. Java Logging API
2. Log4j
3. Logback
4. Tinylog

5. SLF4j
6. JCL (Jakarta Commons Logging) - old name was Apache Commons Logging
   (5th and 6th are logging wrappers)


Application
execute
track the information or issues or messages etc
(Logging Process)
log file


Application
messages with levels
Logger
Levels
Filters
generate log records
Formatter
Handler
Levels
Filters
CMD
log file
database

```
hello 1
Aug 05, 2023 6:32:12 PM in.sp.logging.Test1 main
SEVERE: Server is not responding
Aug 05, 2023 6:32:12 PM in.sp.logging.Test1 main
WARNING: 3 invalid login attempts
Aug 05, 2023 6:32:12 PM in.sp.logging.Test1 main
INFO: this is information message
hello 2
```

```
main method starts
m1 method starts
Aug 05, 2023 6:36:48 PM in.sp.logging.Test2 m1
SEVERE: m1 method severe message
Aug 05, 2023 6:36:48 PM in.sp.logging.Test2 m1
WARNING: m1 method warning message
Aug 05, 2023 6:36:48 PM in.sp.logging.Test2 m1
INFO: m1 method info message
m1 method ends
Aug 05, 2023 6:36:48 PM in.sp.logging.Test2 main
SEVERE: main method severe message
Aug 05, 2023 6:36:48 PM in.sp.logging.Test2 main
WARNING: main method warning message
Aug 05, 2023 6:36:48 PM in.sp.logging.Test2 main
INFO: main method info message
main method ends
```

```java
try
{
    System.out.println("hello 1");

    FileHandler fileHandler = new FileHandler("D:\\mydemo.log");
    SimpleFormatter simpleFormatter = new SimpleFormatter();
    fileHandler.setFormatter(simpleFormatter);

    Logger logger = Logger.getLogger("Test1");
    logger.addHandler(fileHandler);

    logger.log(Level.SEVERE, "Server is not responding");
    logger.log(Level.WARNING, "3 invalid login attempts");
    logger.log(Level.INFO, "this is information message");

    System.out.println("hello 2");
}
catch(Exception e)
{
    e.printStackTrace();
}
```

mydemo - Notepad

File Edit Format View Help

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE log SYSTEM "logger.dtd">
<log>
<record>
  <date>2023-08-05T13:10:21.279306100Z</date>
  <millis>1691241021279</millis>
  <nanos>306100</nanos>
  <sequence>0</sequence>
  <logger>Test1</logger>
  <level>SEVERE</level>
  <class>in.sp.logging.Test3</class>
  <method>main</method>
  <thread>1</thread>
  <message>Server is not responding</message>
</record>
<record>
  <date>2023-08-05T13:10:21.440867300Z</date>
  <millis>1691241021440</millis>
  <nanos>867300</nanos>
  <sequence>1</sequence>
```

mydemo - Notepad

File Edit Format View Help

Aug 05, 2023 6:41:48 PM in.sp.logging.Test3 main
SEVERE: Server is not responding
Aug 05, 2023 6:41:48 PM in.sp.logging.Test3 main
WARNING: 3 invalid login attempts
Aug 05, 2023 6:41:48 PM in.sp.logging.Test3 main
INFO: this is information message

```
Enter no1
10
Aug 05, 2023 6:50:50 PM in.sp.calc.Calculator1 main
INFO: user has provided no1 i.e. 10
Enter no2
20
Aug 05, 2023 6:50:52 PM in.sp.calc.Calculator1 main
INFO: user has provided no2 i.e. 20
Select Any One Symbol (+, -, *, /)
+
Aug 05, 2023 6:50:54 PM in.sp.calc.Calculator1 main
INFO: user has provided symbol i.e. +
Sum is : 30
Aug 05, 2023 6:50:54 PM in.sp.calc.Calculator1 main
INFO: result is 30
```

```
Logger logger = Logger.getLogger("Calculator1");

Scanner scanner = new Scanner(System.in);

System.out.println("Enter no1");
int no1 = scanner.nextInt();
logger.log(Level.INFO, "user has provided no1 i.e. "+no1);

System.out.println("Enter no2");
int no2 = scanner.nextInt();
logger.log(Level.INFO, "user has provided no2 i.e. "+no2);

System.out.println("Select Any One Symbol (+, -, *, /)");
String symbol = scanner.next();
logger.log(Level.INFO, "user has provided symbol i.e. "+symbol);

int res;
```

```
Logger.setLevel(Level.SEVERE);
```

```
1. Java Logging API :-
    -> Java Logging API was introduced in JDK 1.4 version
    -> It is pre-defined in JDK so no need to download any jar file or provide dependency
    -> This API is present in "java.util.logging" package
    -> Components of logging :-
        = There are 2 components in logging :-
            i. Logger
            ii. Handler

-----------------------------------------
```

---

i. Logger :-

-> Logger is an object in logging framework that you use to emit the log messages

-> Some levels of Logger are :-

1. SEVERE (highest value)
2. WARNING
3. INFO
4. CONFIG
5. FINE
6. FINER
7. FINEST (lowest value)

ii. Handler :-

-> Handler is also an object that listens the messages at or above a specified minimum security level

-> Handler will take the messages and post it to the provided medium like console or file or database

-> There are 5 handlers :-

1. ConsoleHandler
2. FileHandler
3. StreamHandler
4. SocketHandler
5. MemoryHandler

=> Log4j API :-
    -> Log4j is an open-source logging API for java
    -> Log4j was introduced in 2001
    -> It is used to store the log details and make it available for tracking the errors or messages or instructions etc
    -> Log4j is fast, reliable and flexible logging API

    -> Components of Log4j :-
        i. Logger
        ii. Appender

---

i. Logger :-
    -> It is an object or component which generates the log messages
    -> Security levels of Logger are :-
        1. OFF (highest value)
        2. Fatal - fatel()
        3. ERROR - error()
        4. WARN - warn()
        5. INFO - info()
        6. DEBUG - debug()
        7. TRACE (lowest value) - trace()

ii. Appender :-

-> It is an object or component which determines where the log messages are sent for storage or display

-> Some Appenders in Log4j are :-

1. ConsoleAppender
2. FileAppender
3. WriterAppender
4. JDBCAppender
5. SocketAppender
6. TelnetAppender
7. SMPTAppender
8. SystlogAppender

-> NOTE : To use Log4j, we have to download one jar file i.e. log4j.*.jar

```
log4j.rootLogger = INFO, CA

log4j.appender.CA = org.apache.log4j.ConsoleAppender
log4j.appender.CA.layout = org.apache.log4j.PatternLayout
log4j.appender.CA.layout.ConversionPattern = %p %d %m%n
```

```
{
    System.out.println("--------- App Starts ---------");

    PropertyConfigurator.configure(System.getProperty("user.dir")+"/src/in/sp/resources/lo

    Logger logger = Logger.getLogger("Main");

    logger.fatal("This is fatal message");
    logger.error("This is error message");
    logger.warn("This is warn message");
    logger.info("This is info message");

    System.out.println("--------- App Stops ---------");
}
```

```
#configurations for FileAppender
log4j.appender.FA = org.apache.log4j.FileAppender
log4j.appender.FA.File = d:/log4jfile.log
log4j.appender.FA.layout = org.apache.log4j.PatternLayout
log4j.appender.FA.layout.ConversionPattern = [%p] - %d{dd/MM/yyyy hh:mm:ss aa} - %m%n
```

```
#configurations for file
log4j.appender.FA = org.apache.log4j.FileAppender
log4j.appender.FA.File = d:/log4jfile.log
log4j.appender.FA.layout = org.apache.log4j.PatternLayout
log4j.appender.FA.layout.ConversionPattern = [%p] - %d{dd/MM/yyyy hh:mm:ss aa} - %m%n

#configurations for console
log4j.appender.CA = org.apache.log4j.ConsoleAppender
log4j.appender.CA.layout = org.apache.log4j.PatternLayout
log4j.appender.CA.layout.ConversionPattern = %p %d %m%n
```

```java
package in.sp.main;

import org.apache.log4j.PropertyConfigurator;

public class Main
{
    public static void main(String[] args)
    {
        System.out.println("--------- App Starts ---------");

        PropertyConfigurator.configure(System.getProperty("user.dir")+"/src/in/sp/resources/l

        Logger logger = Logger.getLogger("Main");

        logger.fatal("This is fatal message");
        logger.error("This is error message");
        logger.warn("This is warn message");
        logger.info("This is info message");

        System.out.println("--------- App Stops ---------");
    }
}
```

Main.java | log4j.properties

```
log4j.rootLogger = INFO, FA

log4j.appender.FA = org.apache.log4j.FileAppender
log4j.appender.FA.File = d:/log4jfile.html
log4j.appender.FA.layout = org.apache.log4j.HTMLLayout
log4j.appender.FA.layout.Title = My Log Messages
```

=> Whenever we create an application, we always want to execute it for different countries having different languages and time zones and cultures

=> For this purpose we use "Internationalization" and "Localization"

=> Some data or information that is different for different countries :-

- = Number
- = Currency
- = Date
- = Time
- = Messages
- = Phone Numbers
- = Address
- etc

```
=> Internationalization :-
    -> It is also known as "I18N"
    -> Internationalization means designing and creating the application (web or software) in such a way
    that makes them easy to adapt for the people of different countries or cultures
    -> It's like making the foundation that can later be customized to fit for the preferences and
    languages of different countries


=> Localization
    -> It is also known as "L10N"
    -> It is like taking the foundation from Internationalization and customzing it for a specific reason
    or language or culture


=> To achieve Internationalization and Localization, java has provided some pre-defined classes :-
    1. Locale
    2. NumberFormat
       DecimalFormat
    3. DateFormat
       SimpleDateFormat
    4. ResourceBundle
       etc
```

```java
Locale locale = Locale.getDefault();

System.out.println("Default locale : "+locale);
System.out.println("Country Name : "+locale.getDisplayCountry());
System.out.println("Country Code : "+locale.getCountry());
System.out.println("Country Langauge : "+locale.getDisplayLanguage());
System.out.println("Language Code : "+locale.getLanguage());

System.out.println("-----------------------------------------------------");

Locale locale2 = new Locale("fr", "FR");
System.out.println("Provided locale : "+locale2);
System.out.println("Country Name : "+locale2.getDisplayCountry());
System.out.println("Country Code : "+locale2.getCountry());
System.out.println("Country Langauge : "+locale2.getDisplayLanguage());
System.out.println("Language Code : "+locale2.getLanguage());
```

```
String[] countryCode_arr = Locale.getISOCountries();
for(String countryCode : countryCode_arr)
{
    System.out.println(countryCode);
}
```

```
String[] countryCode_arr = Locale.getISOCountries();
for(String countryCode : countryCode_arr)
{
    Locale locale = new Locale("", countryCode);
    System.out.println(countryCode+" -> "+locale.getDisplayCountry());
}
```

```
String[] langaugeCode_arr = Locale.getISOLanguages();
for(String languageCode : langaugeCode_arr)
{
    Locale locale = new Locale(languageCode);
    System.out.println(languageCode+" -> "+locale.getDisplayLanguage());
}
```

```java
//Locale mylocale = Locale.JAPANESE;
//Locale mylocale = Locale.CHINESE;
Locale mylocale = new Locale("hi");

String[] countryCode_Arr = Locale.getISOCountries();
for(String countryCode : countryCode_Arr)
{
    Locale locale = new Locale("", countryCode);
    System.out.println(countryCode+" -> "+locale.getDisplayCountry()+" - "+locale.getDisplayCountry(myl
}
```

```java
int no1 = 123456789;
double no2 = 98765.43210;

System.out.println("--------------Below is Indian Format-----------------");
Locale locale1 = Locale.getDefault();
NumberFormat nf1 = NumberFormat.getInstance(locale1);
System.out.println(nf1.format(no1));
System.out.println(nf1.format(no2));


System.out.println("--------------Below is France Format-----------------");
Locale locale2 = new Locale("fr", "FR");
NumberFormat nf2 = NumberFormat.getInstance(locale2);
System.out.println(nf2.format(no1));
System.out.println(nf2.format(no2));
```

```
Locale locale1 = Locale.getDefault();
Currency currency = Currency.getInstance(locale1);
System.out.println(currency.getSymbol()+" -> "+currency.getDisplayName());
```

```java
int no1 = 123456789;
double no2 = 98765.43210;

Locale locale1 = Locale.getDefault();

NumberFormat nf1 = NumberFormat.getCurrencyInstance(locale1);
System.out.println(nf1.format(no1));
System.out.println(nf1.format(no2));
```

```java
int no1 = 123456789;
double no2 = 9876.543210;

//String pattern = "######.#####";
//String pattern = "######.##";
String pattern = "##,##,##.#####";

DecimalFormat dm1 = new DecimalFormat(pattern);
System.out.println(dm1.format(no1));
System.out.println(dm1.format(no2));
```

```java
Date date = new Date();

//System.out.println(date);

System.out.println("----------Below is Indian Format----------");
Locale locale1 = new Locale("en", "IN");
DateFormat df1 = DateFormat.getDateInstance(0, locale1);
System.out.println(df1.format(date));
```

```java
Date date = new Date();

//System.out.println(date);

//String pattern = "dd/MM/yyyy";
//String pattern = "dd/MMM/yyyy";
//String pattern = "dd/MMM/yy";
//String pattern = "dd MM yyyy";
//String pattern = "dd MMM yyyy";
//String pattern = "dd-MM-yyyy";
String pattern = "dd-MMM-yyyy HH:mm:ss";

SimpleDateFormat sdf1 = new SimpleDateFormat(pattern);
System.out.println(sdf1.format(date));
```

```
Choose any one option from below :-
1. English - US
2. Hindi - India
---------------------------------------
1
```

```
LoginFrame()
{
    JFrame jf = new JFrame();
    jf.setSize(600, 600);
    jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    jf.setLayout(null);

    JLabel jl_login_title = new JLabel("Login Here");
    jl_login_title.setBounds(250, 50, 100, 30);
    jf.add(jl_login_title);

    JLabel jl_email_title = new JLabel("Enter Email");
    jl_email_title.setBounds(60, 150, 100, 30);
    jf.add(jl_email_title);

    JTextField jt_email = new JTextField();
    jt_email.setBounds(60, 200, 300, 40);
    jf.add(jt_email);

    jf.setVisible(true);
```

```
    JTextField jt_email = new JTextField();
    jt_email.setBounds(100, 180, 300, 40);
    jf.add(jt_email);

    JLabel jl_pass_title = new JLabel("Enter Passsword :");
    jl_pass_title.setBounds(100, 270, 130, 30);
    jf.add(jl_pass_title);

    JTextField jt_pass = new JTextField();
    jt_pass.setBounds(100, 300, 300, 40);
    jf.add(jt_pass);

    JButton jb_login = new JButton("Login");
    jb_login.setBounds(200, 400, 100, 40);
    jf.add(jb_login);

    jf.setVisible(true);
}
public static void main(String[] args)
{
```

=> ResourceBundle :-
    -> It is an abstract class which is present in "java.util" package
    -> It is used to achieve internationalization wrt messages (string) or images etc

    -> NOTE : We have to use properties file

=> Naming convention of "properties file" :-

    -> basename_languageCode_countryCode_systemVarient.properties

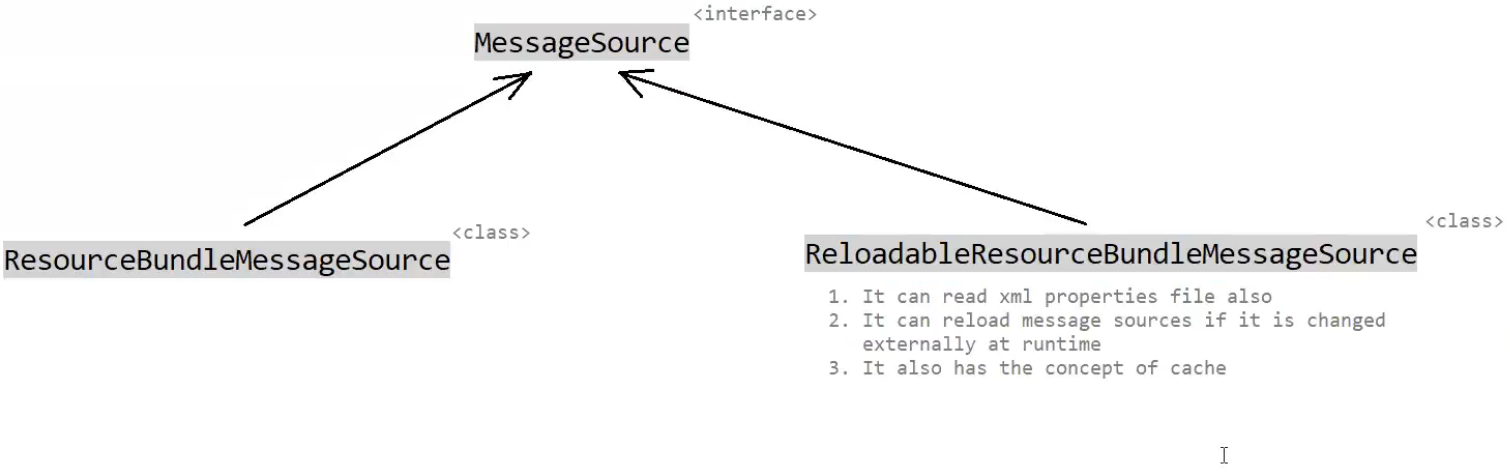
    -> For example :-
        MessageBundle_en_US.properties
        MessageBundle_hi_IN.properties

==========

```xml
<!-- bean definitions here -->
<bean class = "org.springframework.context.support.ResourceBundleMessageSource"
            id = "mgS">
            <!--ye inbuilt method hai usme set kar rahe hai value  -->
        <property name = "basename" value = "me/resources/MessageBundle"/>
</bean>

<bean class = "me.beans.Student" id= "std">
<!-- then uska reference hum yaha set kar rahe hai-->
    <property name ="messageSource" ref ="mgS"/>
</bean>
```

```java
blic class Student {
  private MessageSource messageSource;

  public void setMessageSource(MessageSource messageSource) {
      this.messageSource = messageSource;
  }

  public  void display() {
      Locale locale = new Locale("en","US");
      String message = messageSource.getMessage("key_message",null, locale);
      System.out.println(message);
  }
```

```java
@Bean
public ResourceBundleMessageSource rsmSrcId()
{
    ResourceBundleMessageSource obj = new ResourceBundleMessageSource();
    obj.setBasename("in/sp/resources/MessageBundle");

    return obj;
}

@Bean
public Student stdId()
{
    Student std = new Student();
    std.setMsgsource(rsmSrcId());

    return std;
}
```


MessageSource
<interface>
ResourceBundleMessageSource
<class>
ReloadableResourceBundleMessageSource
<class>
1. It can read xml properties file also
2. It can reload message sources if it is changed externally at runtime
3. It also has the concept of cache

```
=> SpEL :-
    -> SpEL stands for "Spring Expression Language"
    -> SpEL is an "Expression Langauge" which allows you to define and evaluate the expressions at runtime
    -> Features supported by SpEL are :-
        1. Operators :-
            = Arithmetic Operators
                -> +, -, *, / etc
            = Relational Operators
                -> ==   or  eq
                -> !=   or  ne
                -> >    or  gt
                -> <    or  lt
                -> >=   or  ge
                -> <=   or  le
            = Logical Operators
                -> &&   or  and
                -> ||   or  or
                -> !    or  not
            = Ternary Operator
                -> variable = conditional-expression ? expression1 : expression2
            = Type Operator
                -> T(ClassName)
```

```
    2. Expressions :-
        = Literal Expressions
        = Method Invocation
        = Constructor Invocation
        = Regular Expressions (RegEx)
        = Class Expressions
        = Templated Expressions
          etc
    3. Accessing arrays, lists, maps etc
    4. Bean References
       etc
-> How to use SpEL :-
    = We can use SpEL by 2 ways :-
        1. Using pre-defined interfaces and classes
        2. #{expression}
```

==========

=> Pre-defined interfaces and classes in SpEL are :-

1. ExpressionParser <interface>
2. SpelExpressionParser <class>

3. Expression <interface>
4. SpelExpression <class>

5. EvaluationContext <interface>
6. StandardEvaluationContext <class>

-> NOTE : Above all interfaces and classes are present in "org.springframework.expression"

1. ExpressionParser :-
    -> It is an interface which is responsible to parse (resolve) a string expression
    -> Method :-
        = parseExpression(-)

2. SpelExpressionParser :-
    -> It is an implemented class for ExpressionParser

---

3. Expression :-
    -> It is an interface which is responsible to evaluate the string expression
    -> Methods :-
        = getValue(-)
        = getValueType()
        = getValueTypeDescriptor()
        = getExpressionString()
          etc

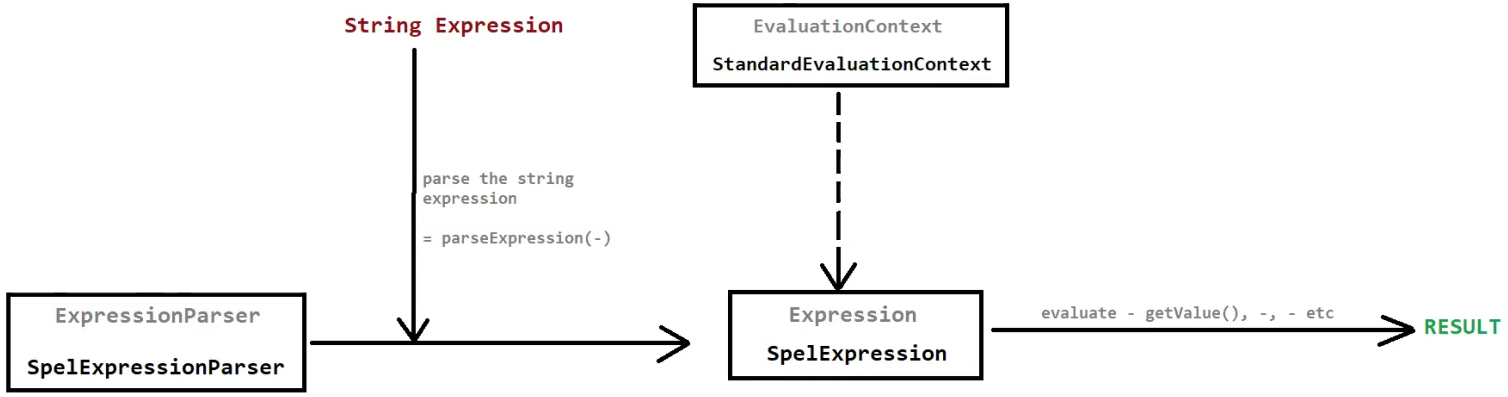
4. SpelExpression :-
    -> It is an implemented class for Expression

```
ExpressionParser parsor = new SpelExpressionParser();

Expression expression = parsor.parseExpression(" 10+20 ");
Object obj =  expression.getValue();
System.out.println(obj);
```

```
ExpressionParser parsor = new SpelExpressionParser();

Expression expression = parsor.parseExpression(" '10+20' ");
Object obj =  expression.getValue();
System.out.println(obj);
```

```
ExpressionParser parsor = new SpelExpressionParser();

Expression expression = parsor.parseExpression(" 10+20 ");
Object obj =  expression.getValue();
System.out.println(obj);


//-----------------------------------------------------------------

ExpressionParser parsor = new SpelExpressionParser();

EvaluationContext context = new StandardEvaluationContext();
context.setVariable("no1", 100);
context.setVariable("no2", 200);

Expression expression = parsor.parseExpression(" #no1 + #no2");
Object obj = expression.getValue(context);
System.out.println(obj);
```

SpEL Workflow



---

3. Expression :-
   -> It is an interface which is responsible to evaluate the string expression
   -> Methods :-
      = getValue(-)
      = getValueType()
      = getValueTypeDescriptor()
      = getExpressionString()
        etc

4. SpelExpression :-
   -> It is an implemented class for Expression

---

5. EvaluationContext :-
   -> It is an interface which is responsible for providing the context in which expressions are evaluated. It holds the variables, functions and other contextual information required for evaluating SpEL expression

6. StandardEvaluationContext :-
   -> It is an implemented class for EvaluationContext

==========

```java
ExpressionParser parsor = new SpelExpressionParser();

Expression expression = parsor.parseExpression(" 10 == 20 ");
System.out.println(expression.getValue());


Expression expression = parsor.parseExpression(" 10 ne 20 ");
System.out.println(expression.getValue());


Expression expression = parsor.parseExpression(" (10 < 20) && (30 < 40) ");
System.out.println(expression.getValue());


Expression expression = parsor.parseExpression(" (10 > 20) ? '111' : '222' ");
System.out.println(expression.getValue());
```

```
ExpressionParser parsor = new SpelExpressionParser();

String str_exp = " 'deepak panwar'.length() ";
Expression expression = parsor.parseExpression(str_exp);
System.out.println(expression.getValue());


String str_exp = " 'deepak panwar'.toUpperCase() ";
Expression expression = parsor.parseExpression(str_exp);
System.out.println(expression.getValue());
```

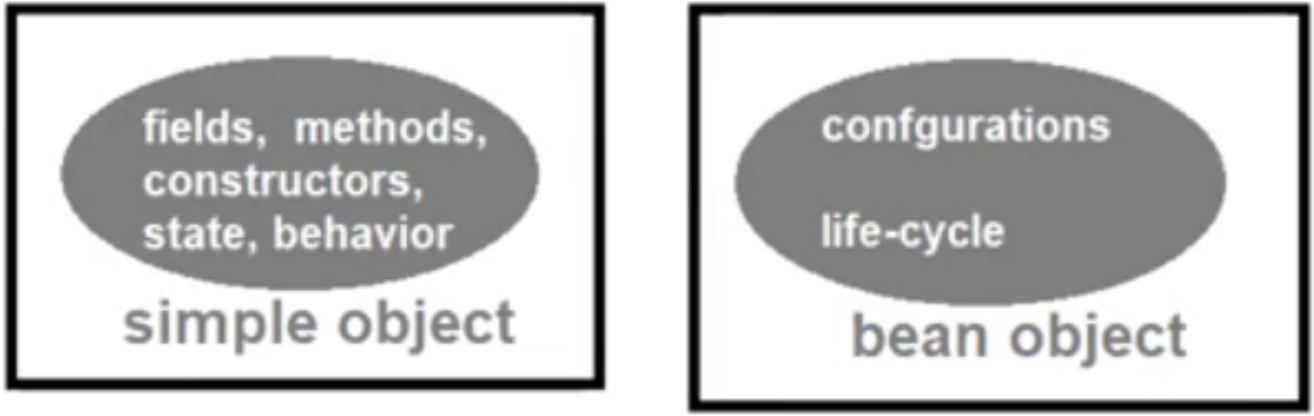
```
ExpressionParser parsor = new SpelExpressionParser();


String str_exp = " 'deepak' matches 'Deepak'  ";
Expression expression = parsor.parseExpression(str_exp);
System.out.println(expression.getValue());


String str_exp = " 'deepak' matches '[a-zA-Z]{5,15}' ";
Expression expression = parsor.parseExpression(str_exp);
System.out.println(expression.getValue());
```

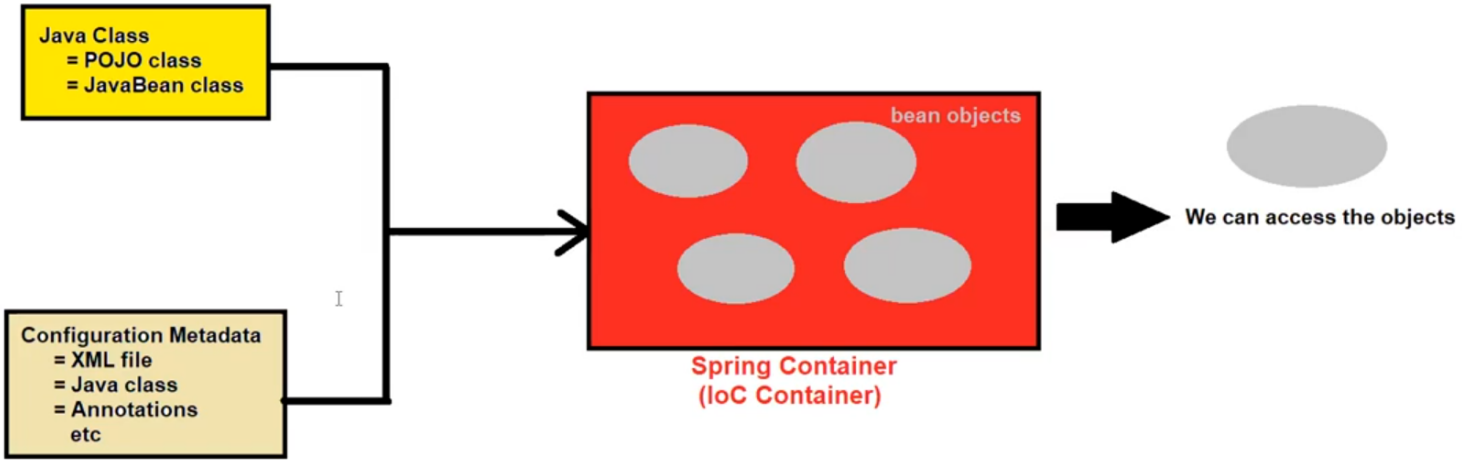
```
ExpressionParser parsor = new SpelExpressionParser();

Expression expression = parsor.parseExpression(" T(in.sp.main1.MyClass).m1() ");
expression.getValue();


Expression expression = parsor.parseExpression(" T(in.sp.main1.MyClass).m3() ");
Object obj = expression.getValue();
System.out.println(obj);


Expression expression = parsor.parseExpression(" T(in.sp.main1.MyClass).m3() ");
int i = (int) expression.getValue();
System.out.println(i);


Expression expression = parsor.parseExpression(" T(in.sp.main1.MyClass).m3() ");
int i = expression.getValue(Integer.class);
System.out.println(i);
```

```
Expression expression = parsor.parseExpression(" new in.sp.main1.MyClass().m2() ");
expression.getValue();
```

=> What is Spring ?
-> Spring is an "Open Source Application Framework" which is used to develop any type of application i.e. Standalone Application or Enterprise Application
-> Spring framework was written by Rod Johnson
-> Spring framework was released under Apache 1.0 licence
-> Spring framework was released in June 2003
-> First production version i.e. 1.0 version was released in March 2004
-> Latest version is 6.x version

=> Advantages of Spring Framework :-
1. Dependency Injection
2. High Level Abstraction and Simplified Development
3. Enhanced Integration & Ecosystem
4. AOP (Aspect Oriented Programming)
5. Easy to test the application
6. Scalabilty and maintability
etc


fields, methods,
constructors,
state, behavior
simple object
confgurations
life-cycle
bean object


Java Class
= POJO class
= JavaBean class
Configuration Metadata
= XML file
= Java class
= Annotations
etc
bean objects
Spring Container
(IoC Container)
We can access the objects

=> What is Spring Container :-
-> Spring container is the "Heart" or "Core Component" of spring framework
-> It is same like :-
= JVM which is used to execute java programs
= Servlet Container or JSP Container which is used to execute Servlet and JSP pages
= EJB Container which provides runtime environment for enterprise beans
-> Responsibilites of Spring Container :-
1. Instantiate bean object
2. Configure bean objects
3. Initialize bean objects
4. Manage bean life-cycle
5. Destroy bean objects
6. Dependency Injection
7. Resource Management
8. AOP (Aspect Oriented Programming)
etc
-> Types of Spring Container :-
1. BeanFactory (old)
2. ApplicationContext (new)
-> Working of Spring Container :-

=> What do you mean by configurations in spring ?

-> Configurations refers to the settings or instructions provided to the spring framework (spring container) to define and manage various aspects of an application

-> There are a lot configurations in spring :-

1. Bean definations
2. Dependency injection
3. Bean life-cycle
4. Bean autowiring
5. Bean post-processing
6. Component scanning
7. Database configurations
8. AOP
9. View resolvers
10. Security configurations
11. Property files

etc

-> In spring, we can provide configurations by :-

1. XML files
2. Java classes
3. Annotations
4. Property files
5. Environment variables
6. Command-line arguments
7. Profiles

etc

=> What is POJO class :-

-> POJO stands for "Plain Old Java Object"

-> It is simple java class which follows some basic conventions for encapsulation, modularity and maintability

-> Syntax :-

```
public class Student
{
    String name;
    public int rollno;
    private int marks;

    //getter and setter methods
}
```

=> What is JavaBean class :-

-> JavaBean class is the class which encapsulates many objects/properties into single unit

-> JavaBean class is a special type of POJO class which follows the following conventions :-

1. Class must be public
2. It must inherit "Serializable" interface
3. It must contain public no-arugment constructor
4. All the properties must be private
5. It should have public getter and setter methods

-> Syntax :-

```
public class Student implements Serializable
{
    public Student(){}

    private String name;
    private int rollno;
    private int marks;

    //public getter and setter methods
}
```

-> NOTE : All JavaBean classes are POJO classes but all POJO classes are not JavaBean classes

=> What is difference between POJO class and JavaBean class :-

1. POJO class is the class whch does not have any restriction
   JavaBean class is the POJO class having some restrictions

2. POJO class may/may not inherit Serializable interface
   JavaBean class must inherit Serializable interface

3. POJO class may/may not contain no-arugment constructor
   JavaBean class must have public no-arugment constructor

4. In POJO class, fields or properties can have any visibility i.e. private, public, default
   In JavaBean class, fields or properties must be private

5. In POJO classes, fields or properties can be accessed by their names
   In JavaBean classes, fields or properties can be accessed only by getter and setter methods

6. POJO class does not have control on members
   JavaBean class have full control on members

7. We cannot use annotations in POJO classes
   We can use annotations in JavaBean classes

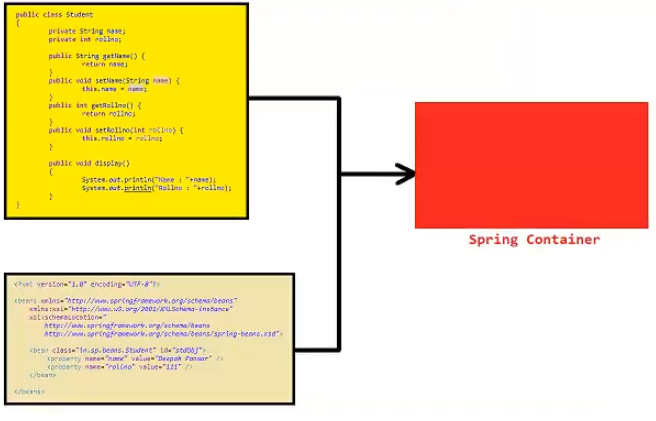
8. We cannot provide ay business logic in POJO classes
   We can provide business logic in JavaBean classes

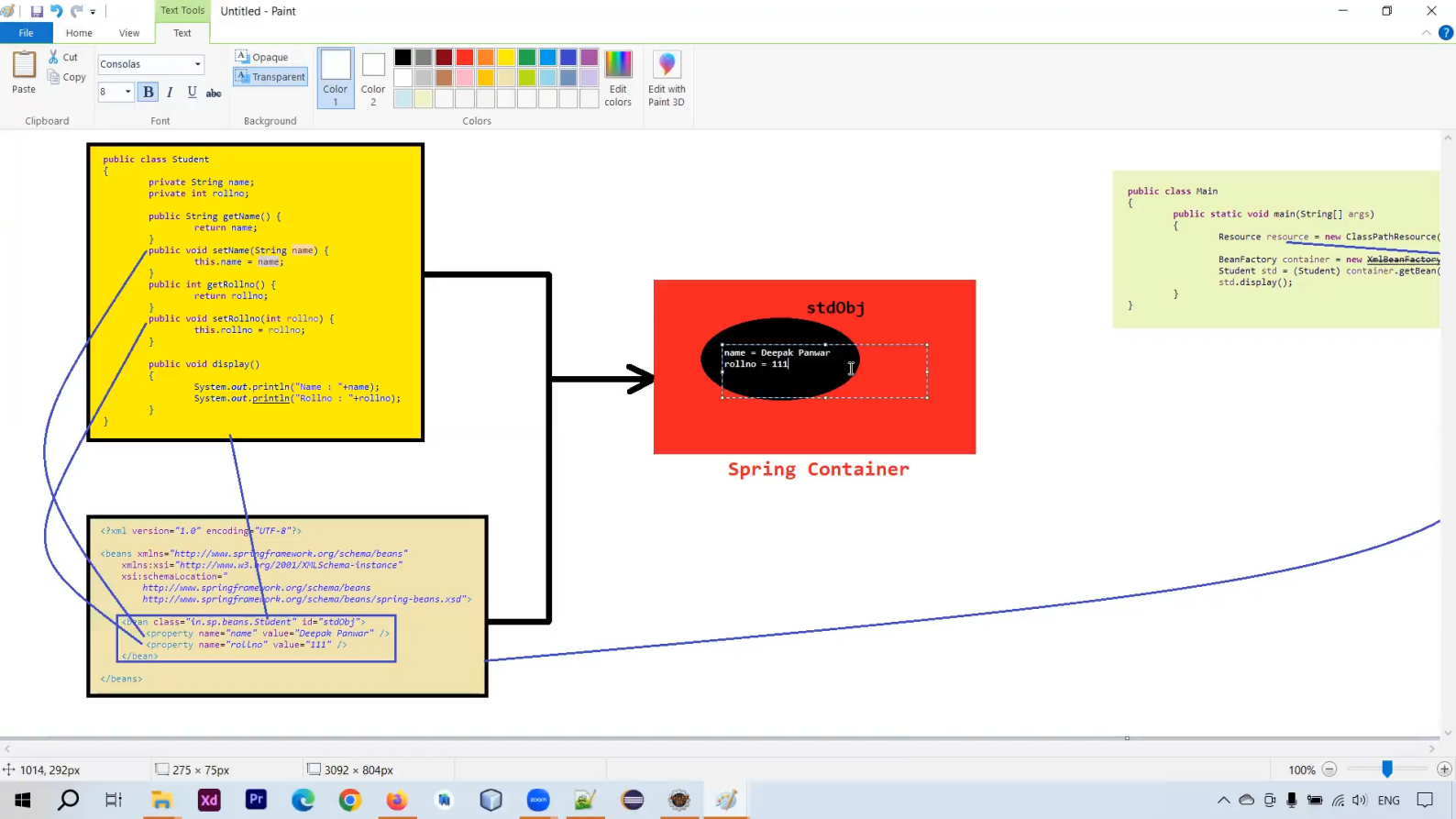
=> What is configuration metadata file :-
-> It is also known as spring configuration file which contains the configuration metadata for our spring application
-> It serves as a central repository to configure beans, dependencies, scopes and other application-specific settings
-> NOTE : If configuration metadata file is an XML file, then name should be "applicationContext.xml"

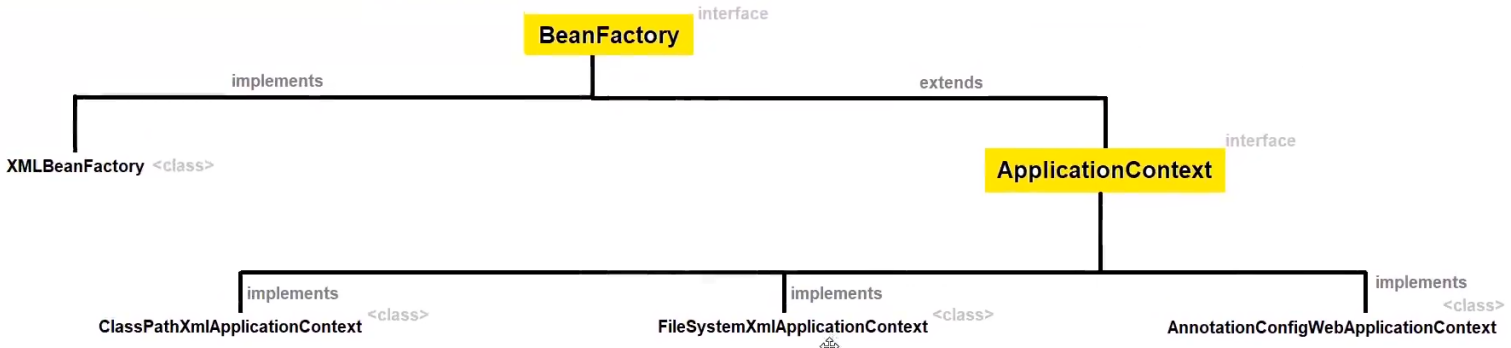
=> Package Name :-
    = Organisation Website : www.google.com
    = com.google.beans

    = www.smartprogramming.in --> in.sp.beans

=> What is Resource ?
    -> Whenever we need to load or read any external file/resource i.e. xml file, text file, properties file, image etc then we use Resource
    -> Resource is an pre-defined interface present in "org.springframework.core.io" package
    -> Spring framework has provided some implemented classes for Resource interface :-
        1. ClassPathResource
        2. URLResource
        3. InputStreamResource
        4. ByteArrayResource
        5. FileSystemResource
           etc


public class Student
{
private String name;
private int rollno;
public String getName() {
return name;
}
public void setName(String name) {
this.name = name;
}
public int getRollno() {
return rollno;
}
public void setRollno(int rollno) {
this.rollno = rollno;
}
public void display()
{
System.out.println("Name : "+name);
System.out.println("Rollno : "+rollno);
}
}
Spring Container
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
<bean class="in.sp.beans.Student" id="std001">
<property name="name" value="Deepak Panwar" />
<property name="rollno" value="111" />
</bean>
</beans>
public class Main
{
public static void main(String[] args)
{
Resource resource = new ClassPathResource("/in/sp/resources/applicationContext.xml");
BeanFactory container = new XmlBeanFactory(resource);
Student std = (Student) container.getBean("std001");
std.display();
}
}


Text Tools
Untitled - Paint
File
Home
View
Text
Cut
Copy
Paste
Consolas
8
Opaque
Transparent
Color 1
Color 2
Edit colors
Edit with Paint 3D
Clipboard
Font
Background
Colors
public class Student
{
private String name;
private int rollno;
public String getName() {
return name;
}
public void setName(String name) {
this.name = name;
}
public int getRollno() {
return rollno;
}
public void setRollno(int rollno) {
this.rollno = rollno;
}
public void display()
{
System.out.println("Name : "+name);
System.out.println("Rollno : "+rollno);
}
}
public class Main
{
public static void main(String[] args)
{
Resource resource = new ClassPathResource(
BeanFactory container = new XmlBeanFactory
Student std = (Student) container.getBean(
std.display();
}
}
stdObj
name = Deepak Panwar
rollno = 111
Spring Container
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
<bean class="in.sp.beans.Student" id="stdObj">
<property name="name" value="Deepak Panwar" />
<property name="rollno" value="111" />
</bean>
</beans>
1014, 292px
275 × 75px
3092 × 804px
100%
ENG


BeanFactory
interface
implements
extends
XMLBeanFactory <class>
ApplicationContext
interface
implements
ClassPathXmlApplicationContext <class>
implements
FileSystemXmlApplicationContext <class>
implements
AnnotationConfigWebApplicationContext <class>

=> What is BeanFactory ?
    -> It is the core interface in the Spring Framework for managing and accessing the beans
    -> It serves as a "Spring Container" that instantiate, configure, manage bean life cycle etc

=> What is ApplicationContext ?
    -> It is the sub-interface of BeanFactory for managing and accessing bean objects
    -> It serves as a "Spring Container" which provides more functionailites as compared to BeanFactory
    -> In simple terms we can say that it is an advanced spring container as compared to BeanFactory

=> Hierarchy of Spring Container :-
    ->

=> Difference between BeanFactory & ApplicationContext ?

1. BeanFactory is the core container or fundamental container
   ApplicationContext is an advanced spring container which provides all the functionailites of BeanFactory container

2. BeanFactory creates the bean object when we call getBean(-) method and thus it is known as lazy instantiation
   ApplicationContext creates the bean object when the container gets started and thus it is known as eager instantiation

3. BeanFactory supports only singleton and prototype scope
   ApplicationContext supports singleton, prototype, request, session scopes

4. BeanFactory does not support I18N functionality
   ApplicationContext supports I18N functionality

5. BeanFactory does not support AOP and ORM
   ApplicationContext supports AOP and ORM

6. BeanFactory does not support annotations
   ApplicationContext supports annotations

7. BeanFactory is suitable for Standalone Applications
   ApplicationContext is suitable for Enterprise Applications

==========

```
<!--
<bean class="in.sp.beans.Student" name="stdObj1, stdObj2">
    <property name="name" value="Deepak Panwar" />
    <property name="rollno" value="222" />
</bean>
-->

<!--
<bean class="in.sp.beans.Student" name="stdObj1 stdObj2">
    <property name="name" value="Deepak Panwar" />
    <property name="rollno" value="222" />
</bean>
-->

<bean class="in.sp.beans.Student" name="stdObj1; stdObj2">
    <property name="name" value="Deepak Panwar" />
    <property name="rollno" value="222" />
</bean>
```

=> What is bean ?

-> Bean are the objects that form the backbone of our spring application which is managed by Spring Container

-> Beans are created with the configuration details/metadata that we provides to spring container using spring configuration file i.e. .xml file or .java file

-> There are some important attributes related to bean objects :-

1. Class
2. Id or Name
3. Property Values
4. Constructor Arguments
5. Scope
6. Initalization and Destruction Callbacks
7. Lazy Initialization
8. Bean Post-Processors
8. Autowiring
9. Profiles

   etc

```
<!--
<bean class="in.sp.beans.Student" name="stdObj1; stdObj2">
    <property name="name" value="Deepak Panwar" />
    <property name="rollno" value="222" />
</bean>
-->

<!--
<bean class="in.sp.beans.Student" name="stdObj1" id="stdObj1">
    <property name="name" value="Deepak Panwar" />
    <property name="rollno" value="222" />
</bean>
-->
```

```
<bean class="in.sp.beans.Student" name="stdObj1">
    <property name="name" value="Deepak Panwar" />
    <property name="rollno" value="222" />
</bean>

<bean class="in.sp.beans.Student" name="stdObj1">
    <property name="name" value="Amit Sharma" />
    <property name="rollno" value="333" />
</bean>
```

=> What is id and name attributes for bean object ?

-> id :-

= It specifies the unique identity of bean object

-> name :-

= It specifies the unique identity of bean object but it is more flexible as compared to id attribute

= Flexibilities that are provided by name attribute are :-

1. We can provide multiple names for one bean object
2. We can separate the multiple bean names by comma(,) or semi-colon(;) or space
3. We can provide same bean object name in name and id attribute
4. We can provide same name to one bean object but same name cannot be provided to multiple bean objects

applicationContext.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean class="in.sp.beans.Student" id="stdObj1">
      <property name="name" value="Deepak Panwar" />
      <property name="rollno" value="222" />
    </bean>

    <bean class="in.sp.beans.Student" id="stdObj2">
      <property name="name" value="Amit Sharma" />
      <property name="rollno" value="333" />
    </bean>

</beans>
```

SpringConfigFile.java

```java
@Configuration
public class SpringConfigFile
{
      @Bean(name = "stdObj1")
      public Student createBeanObj1()
      {
            Student std = new Student();

            std.setName("Deepesh Panwar");
            std.setRollno(444);

            return std;
      }

      @Bean(name = "stdObj2")
      public Student createBeanObj2()
      {
            Student std = new Student();

            std.setName("Kamal Sharma");
            std.setRollno(555);

            return std;
      }
}
```

=> Java Based Configuration :-
-> Before Spring 3.0 version, it was compulsory to provide spring configurations metadata by using xml file. But from spring 3.0 version, its not compulsory to create xml file. We can also provide configurations metadata by using java class
-> How to achieve java based configurations :-
1. Create java class i.e. configuration class and mark it as @Configuration annotation
2. Create one or more methods (which returns bean object) and mark it as @Bean annotation
3. Create object of AnnotationConfigApplicationContext class and access the bean object

-> NOTE : If we dont provide bean object name manually then bean name will be same as that of method name. If we want to declare bean name manually then we can use @Bean(name = "beanObjectName")

=> What is @Configuration annotation ?
-> @Configuration annotation is used with class
-> When spring container starts, it will check all the java classes marked with @Configuration. Then it will load the class into memory and process them to create bean defination/configurations

=> What is @Bean annotation ?
-> @Bean annotation is used with methods
-> @Bean methods are responsible to create and configure bean objects.
-> When spring container starts, it will invokes each @Bean method and create the bean objects
-> By default bean object name is same as method name but if we want to change the bean object name then we can use name attribute i.e. @Bean(name = "beanObjectName")

```
5
6 @Component("stdObj")
7 public class Student
8 {
```

=> @Component :-
    -> It is also known as "stereotype annotation"
    -> It is used to mark the class as a spring-managed component. The spring container is responsible for creating, configuring and managing the components inluding their life-cycle, dependency injection etc
    -> By default @Component scope is "singleton scope"

=> Some examples of spring-managed components are :-
    1. @Configuration
    2. @Bean
    3. @Component
        = @Controller
        = @Service
        = @Repository
    4. @Autowired
    5. @Aspect
       etc

=> Different ways to create bean objects and property configuration :-

1. XML file

```
<bean class="fully qualified JavaBean class name" id="beanId">
    <property name="property_name" value="property_value" />
    <property name="property_name" value="property_value" />
</bean>
```

2. Java class :-

```
@Configuration
class JavaConfigFile
{
    public JavaBean m1()
    {
        JavaBean obj = new JavaBean();

        obj.setXXX(-);
        obj.setXXX(-);

        return obj;
    }
}
```

3. Annotations :-

```
public class JavaBean
```

3. Annotations :-

```
@Component
public class JavaBean
{
    @Value("--")
    private String property_name;
    --
}
```

NOTE : we have to either register the JavaBean class or scan the packages

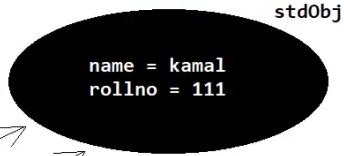
=> Bean Scope :-
    -> Bean Scope defines the visibility or accessability of that bean in the context we use it
    -> We can provide bean scope by using "scope attribute" or "@Scope annotation"
    -> There are total 7 scopes :-
        1. "singleton" scope
        2. "prototype" scope

        3. "request" scope
        4. "session" scope
        5. "globalSession" scope
        6. "application" scope
        7. "webSocket" scope

    -> NOTE : By default, beans are singleton scope

=> "singleton" scope :-
    -> It is the default scope of bean object
    -> In this scope only one instance will be created for a single bean defination and that same object will be shared for each request made for that bean using getBean(-) method
    -> Program

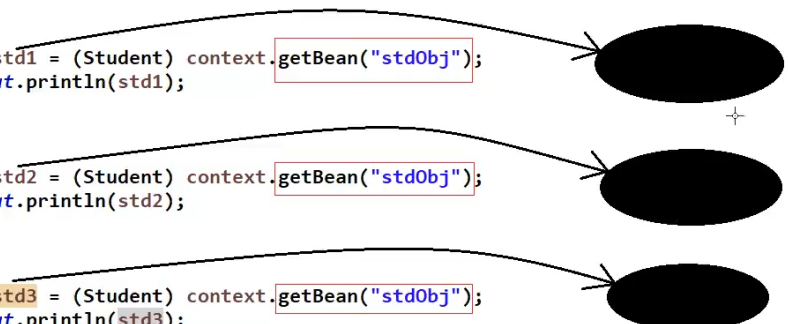
```xml
<bean class="in.sp.beans.Student" id="stdObj">
    <property name="name" value="Kamal" />
    <property name="rollno" value="111" />
</bean>
```




```java
ApplicationContext context = new ClassPathXmlApplicationContext("/in/sp/resources/applicationContext.xml");


Student std1 = (Student) context.getBean("stdObj");
System.out.println(std1);


Student std2 = (Student) context.getBean("stdObj");
System.out.println(std2);


Student std3 = (Student) context.getBean("stdObj");
System.out.println(std3);
```

```xml
<bean class="in.sp.beans.Student" id="stdObj" scope="prototype">
    <property name="name" value="Kamal" />
    <property name="rollno" value="111" />
</bean>
```

```
ApplicationContext context = new ClassPathXmlApplicationContext("/in/sp/resources/applicationContext.xml");


Student std1 = (Student) context.getBean("stdObj");
System.out.println(std1);


Student std2 = (Student) context.getBean("stdObj");
System.out.println(std2);


Student std3 = (Student) context.getBean("stdObj");
System.out.println(std3);
```



=> "prototype" scope :-
    -> In this scope a new instance is created for a single bean defination and the new object will be shared for each request made for that bean using getBean(-) method
    -> Program

=> @Component :-
    -> It is also known as "stereotype annotation"
    -> It is used to mark the class as a spring-managed component. The spring container is responsible for creating, configuring and managing the components inluding their life-cycle, dependency injection etc
    -> By default @Component scope is "singleton scope"

=> Some examples of spring-managed components are :-
    1. @Configuration
    2. @Bean
    3. @Component
        = @Controller
        = @Service
        = @Repository
    4. @Autowired
    5. @Aspect
       etc

=> Different ways to create bean objects and property configuration :-

1. XML file

```
<bean class="fully qualified JavaBean class name" id="beanId">
    <property name="property_name" value="property_value" />
    <property name="property_name" value="property_value" />
</bean>
```

2. Java class :-

```
@Configuration
class JavaConfigFile
{
    public JavaBean m1()
    {
        JavaBean obj = new JavaBean();

        obj.setXXX(-);
        obj.setXXX(-);

        return obj;
    }
}
```

3. Annotations :-

```
public class JavaBean
```

3. Annotations :-

```
@Component
public class JavaBean
{
    @Value("--")
    private String property_name;
    --
}
```

NOTE : we have to either register the JavaBean class or scan the packages

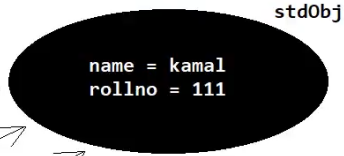
=> Bean Scope :-
    -> Bean Scope defines the visibility or accessability of that bean in the context we use it
    -> We can provide bean scope by using "scope attribute" or "@Scope annotation"
    -> There are total 7 scopes :-
        1. "singleton" scope
        2. "prototype" scope

        3. "request" scope
        4. "session" scope
        5. "globalSession" scope
        6. "application" scope
        7. "webSocket" scope

    -> NOTE : By default, beans are singleton scope

=> "singleton" scope :-
    -> It is the default scope of bean object
    -> In this scope only one instance will be created for a single bean defination and that same object will be shared for each request made for that bean using getBean(-) method
    -> Program

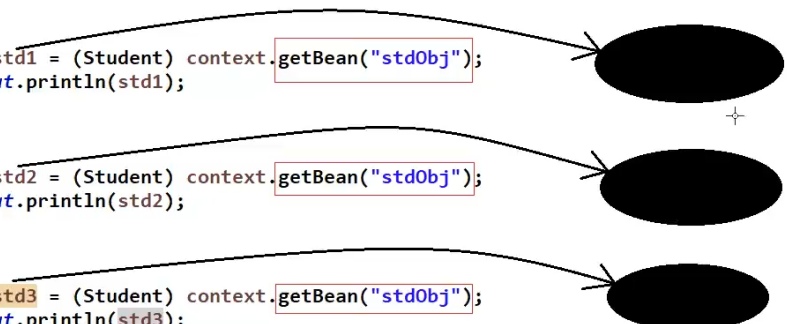
```xml
<bean class="in.sp.beans.Student" id="stdObj">
    <property name="name" value="Kamal" />
    <property name="rollno" value="111" />
</bean>
```




```java
ApplicationContext context = new ClassPathXmlApplicationContext("/in/sp/resources/applicationContext.xml");


Student std1 = (Student) context.getBean("stdObj");
System.out.println(std1);


Student std2 = (Student) context.getBean("stdObj");
System.out.println(std2);


Student std3 = (Student) context.getBean("stdObj");
System.out.println(std3);
```

```xml
<bean class="in.sp.beans.Student" id="stdObj" scope="prototype">
    <property name="name" value="Kamal" />
    <property name="rollno" value="111" />
</bean>
```

```
ApplicationContext context = new ClassPathXmlApplicationContext("/in/sp/resources/applicationContext.xml");


Student std1 = (Student) context.getBean("stdObj");
System.out.println(std1);


Student std2 = (Student) context.getBean("stdObj");
System.out.println(std2);


Student std3 = (Student) context.getBean("stdObj");
System.out.println(std3);
```



=> "prototype" scope :-
    -> In this scope a new instance is created for a single bean defination and the new object will be shared for each request made for that bean using getBean(-) method
    -> Program

=> Bean Life Cycle :-
    1. Loading Bean Definations
    2. Bean Instantiation
    3. Bean Initialization
    4. Bean Usage
    5. Bean Destruction

---

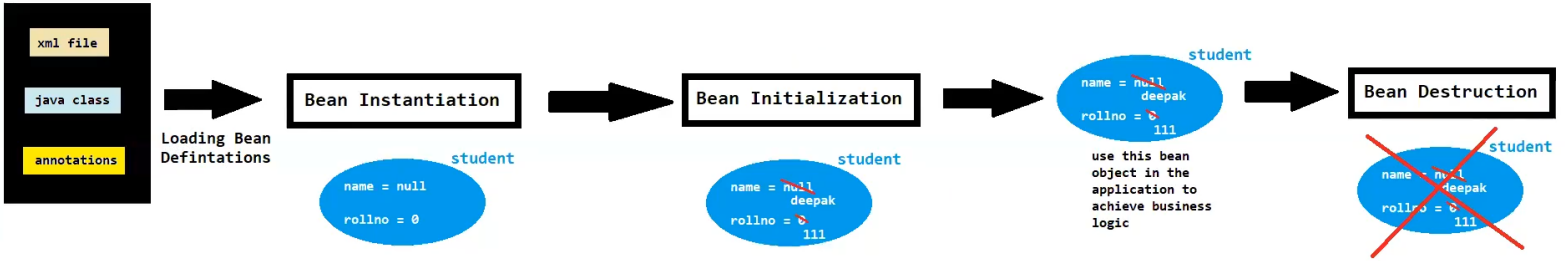
1. Loading Bean Definations :-
    -> Bean definations are the configurations (blueprint or settings) that defines how bean object should be created. It includes the information about the class to instantiate, property configurations, dependency injection and other configurations
    -> Bean definations can be provided by xml file or java class or annotations
    -> It is the process of reading and parsing the configuration files to create bean definations for the beans that will be managed by the spring container

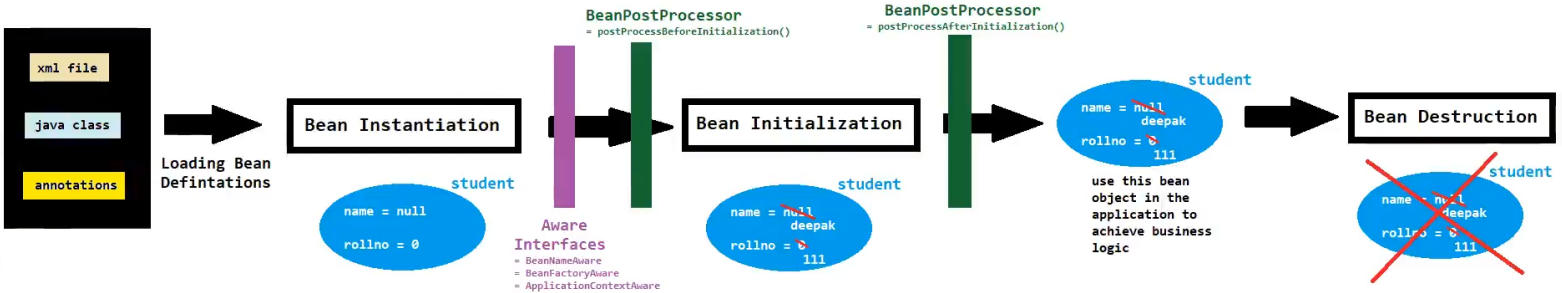
2. Bean Instantiation :-
   -> In this phase, spring container will create an instance of the bean based on it bean definations
   -> How bean objects are created ?
      a. using default constructor or no-argument constructor
      b. using static factory method
      c. using instance factory method
   -> In this phase, bean objects are initialized with default values based on the data types of the properties in the JavaBean class
   -> In this phase, the container also injects the required dependencies into the bean object by any following way :-
      a. Setter method DI
      b. Constructor DI

3. Bean Initialization :-
   -> In this phase bean object is initialized by its original values
   -> How bean objects are initialized ?
      a. using property tags
      b. using explicit ways
         i. using custom init() method
         ii. using afterPropertiesSet() method of InitializingBean callback interface
         iii. using @PostConstruct annotation

3. Bean Initialization :-
   -> In this phase bean object is initialized by its original values
   -> How bean objects are initialized ?
      a. using property tags
      b. using explicit ways
         i. using custom init() method
         ii. using afterPropertiesSet() method of InitializingBean callback interface
         iii. using @PostConstruct annotation (jar file is needed javax.annotation-api-xxx.jar)

4. Bean Usage :-
   -> Once the bean is fully initialized, it is ready to be used in our application.
   -> Beans can be retrieved from the spring container and can be used for business logic in our application


xml file
java class
annotations
Loading Bean Defintations
Bean Instantiation
student
name = null
rollno = 0
Bean Initialization
student
name = null
deepak
rollno = 0
111
student
name = null
deepak
rollno = 0
111
use this bean object in the application to achieve business logic
Bean Destruction
student
name = null
deepak
rollno = 0
111


xml file
java class
annotations
Loading Bean Defintations
Bean Instantiation
student
name = null
rollno = 0
BeanPostProcessor
= postProcessBeforeInitialization()
Aware Interfaces
= BeanNameAware
= BeanFactoryAware
= ApplicationContextAware
Bean Initialization
student
name = null
deepak
rollno = 0
111
BeanPostProcessor
= postProcessAfterInitialization()
student
name = null
deepak
rollno = 0
111
use this bean object in the application to achieve business logic
Bean Destruction
student
name = null
deepak
rollno = 0
111

Student.java | applicationContext.xml | student.properties

```
1 student.name=Deepak
2 student.rollno=101
3 student.subjmarks.C=87
4 student.subjmarks.Cpp=83
5 student.subjmarks.Java=97
6 subject.subjmarks.Python=91
```

```
6
7    <bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
8        <property name="location" value="/in/sp/resources/student.properties" />
9    </bean>
10
11   <bean class="in.sp.beans.Student" id="stdId">
12       <property name="name" value="${student.name}" />
13       <property name="rollno" value="${student.rollno}" />
14       <property name="subjmarks">
15           <map>
16               <entry key="C" value="${student.subjmarks.C}" />
17               <entry key="C++" value="${student.subjmarks.Cpp}" />
18               <entry key="Java" value="${student.subjmarks.Java}" />
19               <entry key="Python" value="${student.subjmarks.Python}" />
20           </map>
21       </property>
```

```
@PropertySource("/in/sp/resources/student.properties")
```

```
@Value("#{${unknownMap : {'C': 1, 'Java': 2}}}")
private Map<String, Integer> subjmarks;
```

=> Property Configurations :-
    -> Property Configurations is the process by which we set the values in the bean properties
    -> We can set the property values either from xml file or java or properties file

=> Spring Configurations :-
    -> It is the process of providing the spring application configurations that how our spring application will work

=> What is properties file ?
    -> A property file is a simple text file commonly used to store configurations data in key-value pair
    -> Advantage of properties file :-
        1. Easy to update or modify the configurations
        2. We dont need to restart the application if we change or modify the configurations
        3. Easy to test the application
           etc

==========

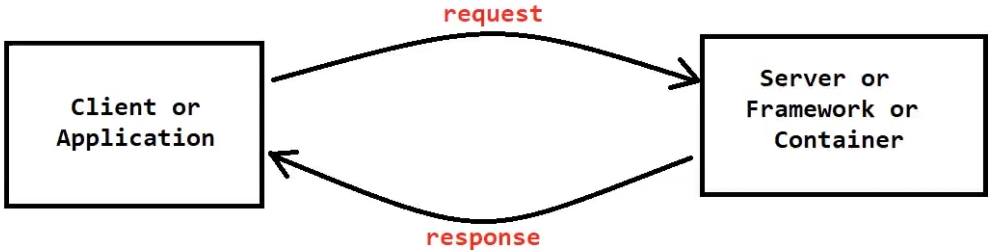
```
<bean class="in.sp.beans.Student" id="stdId">
    <constructor-arg value="Deepak" />
    <constructor-arg value="111" />
    <constructor-arg value="94.8" />
</bean>
```

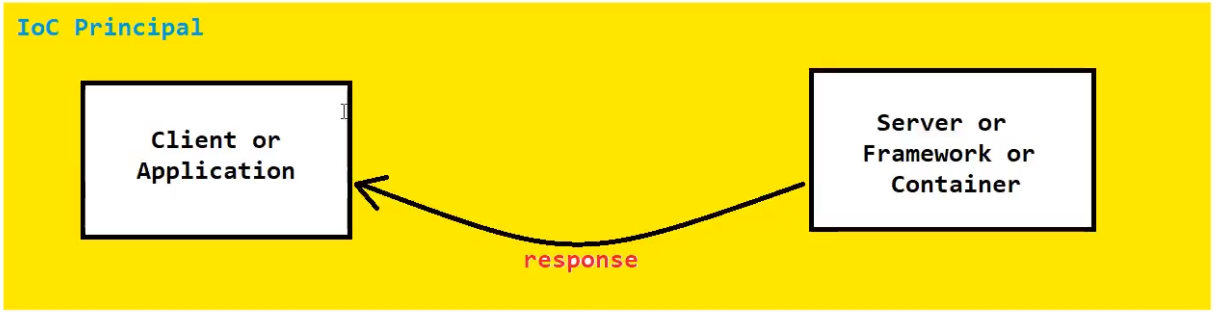
```
<bean class="in.sp.beans.Student" id="stdId">
    <constructor-arg value="111" index="1" />
    <constructor-arg value="Deepak" />
    <constructor-arg value="94.8" />
</bean>
```

```
<bean class="in.sp.beans.Student" id="stdId">
    <constructor-arg value="111" type="int" />
    <constructor-arg value="94.8" type="float" />
    <constructor-arg value="Deepak" />
</bean>
```

=> Property Configurations :-
    -> Property Configurations is the process by which we set the values in the bean properties
    -> We can set the property values either from xml file or java or properties file

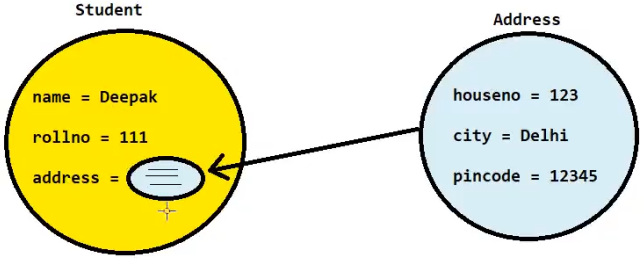
    -> We can provide the property configuration in xml file by 2 ways :-
        1. By using <property> tag
        2. By using <constructor-arg> tag


request
Client or Application
Server or Framework or Container
response


IoC Principal
Client or Application
Server or Framework or Container
response

=> Inversion Of Control (IoC) :-
-> Inversion of Control is a design pattern/principal that focus on inverting the conrol flow of an application
-> It shifts the responsibility of managing the flow of execution and the lifecycle of objects from application itself to external entity i.e. framework or container
-> It identifies the client required dependencies or services and then it will create and inject the required dependencies or service to the application without client request
-> Spring Container works on the basis of IoC principal and thus it is also known as IoC Container
-> Advantages of IoC principal :-
    1. Classes are loosly coupled
    2. Modularity can be achieved
    3. Easier to test and maintain the application
       etc
-> In spring IoC principal can be achieved by following :-
    1. Dependency Injection (DI)
    2. Service Locator
    3. Contextualized Lookup
    4. Template Method Design Pattern
    5. Event Based IoC
       etc
-> NOTE : From above, only DI is most commonly IoC principal used in spring

```
class Student
{
    private String name;
    priavte int rollno;
    private Address address;

    //getter and setter methods

    //other methods
}
```




```
class Address
{
    private int houseno;
    private String city;
    private int pincode;

    //getter and setter methods

    //other methods
}
```

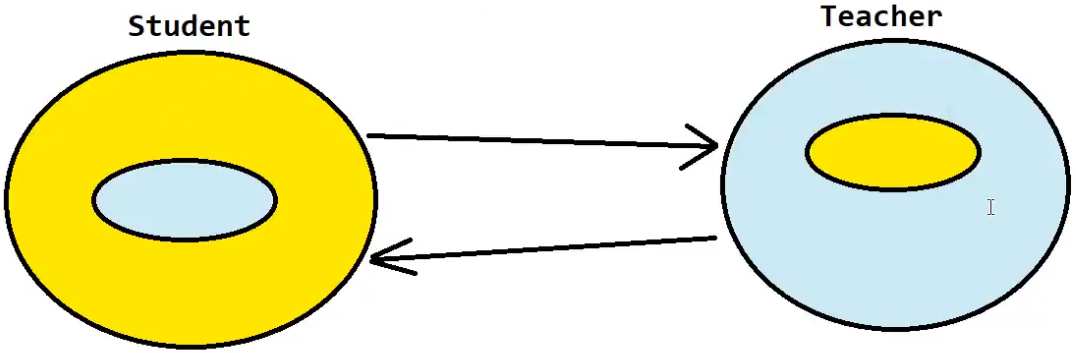
```
<!-- bean definitions here -->
 <bean class ="me.beans.Address" id="addId">
    <property name = "addName" value = "Shukhi Bhawan"/>
    <property name = "houseNo" value = "012"/>
    <property name = "city" value="Dhanbad"/>
</bean>
<bean class ="me.beans.Student" id="stdId">
    <property name = "name" value = "Piyush"/>
    <property name = "id" value = "95"/>
    <property name = "address" ref="addId"/>
</bean>
```

=> Dependency Injection (DI) :-
    -> Dependency Injection is a design pattern that is used to implement IoC principal
    -> Dependency Injection main functionality is to "inject" one object into another object
    -> How to achieve DI in Spring Configuration File (xml) :-
        = We can achieve DI by 2 ways :-
            1. Setter Method DI
            2. Constructor DI

==========

=> What is difference between Setter Method DI and Constructor DI :-
    1. How dependency is injected :-
        = Setter Method DI uses setter methods i.e. setXXX() method to inject the dependency
        = Constructor DI uses constructor to inject the dependency
    2. Readability :-
        = Setter Method DI has more readability because we have to provide the property name and its value
        = Constructor DI has less readability because we dont provide the property name with value
    3. Partial Dependency :-
        = Partial Dependency is possible in case of Setter Method DI
        = Partial Dependency is not possible in case of Constructor DI
    4. Circular DI :-
        = We can achieve Circular DI using Setter Method DI
        = We cannot achieve Circular DI using Constructor DI

Circular DI



```
17
18 !--this is the alternative and basically easier way to set the data bus upar mein ek
19 chema paste krna hai aur kuch nhi  -->
20
21 !--<bean class = "me.beans.Student" id = "std"
22     p:name="piyush"
23     p:rollno="233"
24     p:marks="92.3"
25 />-->
26
```

```
<bean class = "me.beans.Address" id = "add"
    p:addName="Sukhi Bhavan"
    p:houseNo="233"
    p:city="Dhanbad"
    />

<!--if there is any class related to any another class then we have to use ref as shown below-
<bean class = "me.beans.Student" id = "std"
    p:name="piyush"
    p:rollno="233"
    p:marks="92.3"
    p:address-ref="add"
    />
```

=> p-namespace :-
-> When we have to inject dependencies in bean object using setter method then we have to provide <property> tag. More dependencies more <property> tag and due to this the code become lengthy which is not good.
-> To solve this problem, spring has provided one feature or shortcut i.e. p-namespace
-> How to use p-namespace :-
1. Provide p-namespace declaration in spring configuration file
xmlns:p="http://www.springframework.org/schema/p"
2. Then we can provide the dependencies in bean tag by p:property_name="value" OR p:property_name-ref="value"

=> c-namespace :-
-> When we have to inject dependencies in bean object using constructor then we have to provide <constructor-arg> tag. More dependencies more <constructor-arg> tags and due to this the code becomes lengthy which is not good
-> To solve this problem, spring has provided one feature or shortcut i.e. c-namespace
-> How to use c-namespace :-
1. Provide c-namespace declaration in spring configuration file
2. Then we can provide the dependencies in bean tag by c:property_name="value" OR c:_indexPosition="value" OR c:property_name-ref="value"

```
==========================================================================================

=> Depedency Injection using Java Configuration File :-
    -> How to achieve DI in Spring Configuration File (java) :-
        = We can achieve DI by 2 ways :-
            1. Setter Method DI
            2. Constructor DI
    -> Programs
```

=> Dependency Injection (DI) :-
    -> DI is the process by which we can inject one bean object into another bean object
    -> DI can be achieved by 2 ways :-
        1. Setter Method DI
        2. Constructor DI
        (NOTE : both these have been done by using xml and java based configuration)

    -> Till now we have achieved DI by explicit ways

=> Autowiring :-
    -> Autowiring is the feature of Spring Framework by which we can achieve DI automatically
    -> Advantage :-
        = It requires less code
    -> Disadvantage :-
        = There is no control of programmer
        = It can be achieved only on non-primitive or user-defined data types (excluding String), not on primitive data types
    -> How can we achieve autowiring :-
        = We can achieve autowiring by 4 ways :-
            1. XML Based Autowiring
            2. Annotation Based Autowiring
            3. Java Based Autowiring
            4. Component Scanning

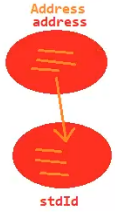
```java
public class Student
{
	private String name;
	private int rollno;
	private Address address;

	//getter and setter methods

	public void display()
	{
		System.out.println("Name : "+name);
		System.out.println("Rollno : "+rollno);
		System.out.println("Address : "+address);
	}
}
```

```java
public class Address
{
	private int houseno;
	private String city;
	private int pincode;

	//getter and setter methods

	@Override
	public String toString()
	{
		return "#"+houseno+", "+city+" - "+pincode;
	}
}
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans ------------>

  <bean class="in.sp.beans.Address" id="address">
    <property name="houseno" value="123" />
    <property name="city" value="Delhi" />
    <property name="pincode" value="12345" />
  </bean>

  <bean class="in.sp.beans.Student" id="stdId" autowire="byType">
    <property name="name" value="Deepak" />
    <property name="rollno" value="111" />
  </bean>

</beans>
```

```
<bean class = "me.beans.Address" id = "address">
    <property name = "addName" value = "Sukhi Bhawan"/>
    <property name = "houseNo" value = "113"/>
    <property name = "city" value = "Dhanbad"/>
</bean>

    <!--yaha pr hum agar bytype use kr rahe hai to jo Address and jo uska
    bean class provide kr rahe hai to hi autowiring achieve ho payega
       achive ho payega-->
```

```java
public class Student
{
	private String name;
	private int rollno;
	private Address address;

	//getter and setter methods

	public void display()
	{
		System.out.println("Name : "+name);
		System.out.println("Rollno : "+rollno);
		System.out.println("Address : "+address);
	}
}
```




```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans ------------>

    <bean class="in.sp.beans.Address" id="address">
      <property name="houseno" value="123" />
      <property name="city" value="Delhi" />
      <property name="pincode" value="12345" />
    </bean>

    <bean class="in.sp.beans.Student" id="stdId" autowire="byType">
      <property name="name" value="Deepak" />
      <property name="rollno" value="111" />
    </bean>

</beans>
```

```java
public class Address
{
	private int houseno;
	private String city;
	private int pincode;

	//getter and setter methods

	@Override
	public String toString()
	{
		return "#"+houseno+", "+city+" - "+pincode;
	}
}
```

```java
public class Main
{
	public static void main(String[] args)
	{
		ApplicationContext context = new ClassPathXmlApplic----("/in/sp/resources/applicationContext.xml");

		Student std = (Student) context.getBean("stdId");
		std.display();
	}
}
```

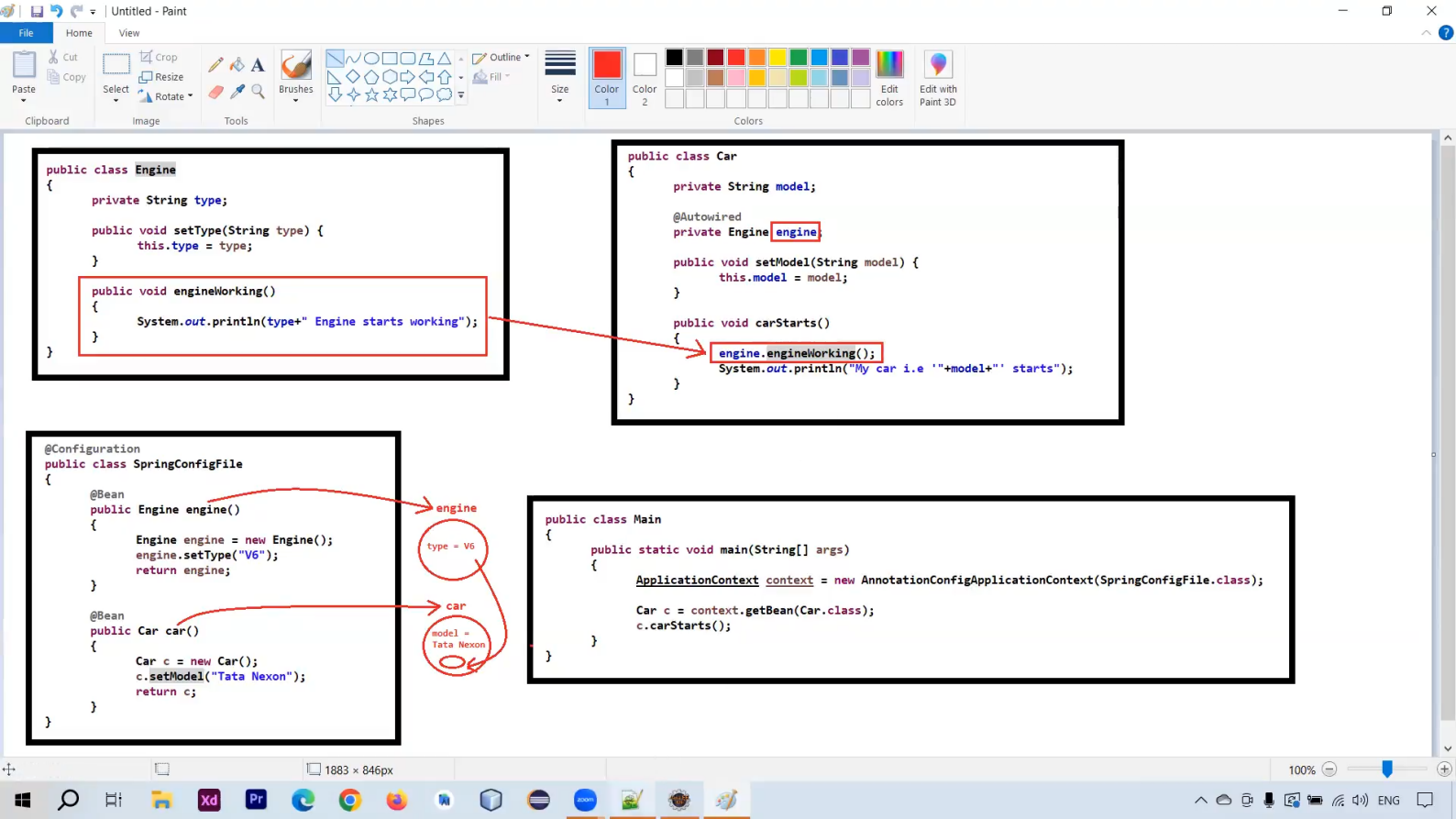
```
    <bean class="in.sp.beans.Address" id="addrId1">
        <property name="houseno" value="123" />
        <property name="city" value="Delhi" />
        <property name="pincode" value="12345" />
    </bean>

    <bean class="in.sp.beans.Address" id="addrId2" autowire-candidate="false">
        <property name="houseno" value="678" />
        <property name="city" value="Mumbai" />
        <property name="pincode" value="67890" />
    </bean>

    <bean class="in.sp.beans.Student" id="stdId" autowire="byType">
        <property name="name" value="Deepak" />
        <property name="rollno" value="111" />
    </bean>
```

```
<bean class="in.sp.beans.Address" id="address">
    <constructor-arg value="123" />
    <constructor-arg value="Delhi" />
    <constructor-arg value="12345" />
</bean>

<bean class="in.sp.beans.Student" id="stdId" autowire="constructor">
    <constructor-arg value="Deepak" index="0" />
    <constructor-arg value="111" index="1" />
</bean>
```

=> XML Based Autowiring :-
-> In case of XML Based Autowiring, we dont need to use "ref" attribute in <property> or <constructor-arg> tag

-> We can achieve XML based autowiring by using "autowire" attribute in <bean> tag i.e.
<bean class="----" id="----" autowire="--modes--">
-> Modes of autowire attribute :-
1. no :-
= It is default autowiring mode
= It simply means that we dont want to achieve autowiring
2. byName
= In this case we will achieve autowiring by matching "property name" of bean object and "bean id" in spring configuration file
= It uses "Setter Method DI" internally
3. byType
= In this case we will achieve autowiring by matching the data-types i.e. "data-types" in bean class should be same as that of "class" in <bean> tag
= It uses "Setter Method DI" internally
= In this case, if we have create multiple bean objects of one class, then which class it will inject, confusion will occur. To remove this confusion we can we one attribute i.e. "autowire-candidate" i.e. autowire-candidate="false". Whenever we will use this attribute with the bean, it will not participate in autowiring
4. constructor
= This is same as that of byType
= It internally use "Constructor DI"
5. autodeduct
= It is depricated from spring 3.x version

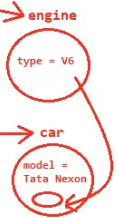
```
@Autowired
@Qualifier("createAddrObj1")
```


Untitled - Paint
File
Home
View
Cut
Copy
Paste
Crop
Resize
Rotate
Select
Brushes
Outline
Fill
Size
Color 1
Color 2
Edit colors
Edit with Paint 3D
Clipboard
Image
Tools
Shapes
Colors
public class Engine
{
private String type;
public void setType(String type) {
this.type = type;
}
public void engineWorking()
{
System.out.println(type+" Engine starts working");
}
}
public class Car
{
private String model;
@Autowired
private Engine engine;
public void setModel(String model) {
this.model = model;
}
public void carStarts()
{
engine.engineWorking();
System.out.println("My car i.e '"+model+"' starts");
}
}
@Configuration
public class SpringConfigFile
{
@Bean
public Engine engine()
{
Engine engine = new Engine();
engine.setType("V6");
return engine;
}
@Bean
public Car car()
{
Car c = new Car();
c.setModel("Tata Nexon");
return c;
}
}
engine
type = V6
car
model = Tata Nexon
public class Main
{
public static void main(String[] args)
{
ApplicationContext context = new AnnotationConfigApplicationContext(SpringConfigFile.class);
Car c = context.getBean(Car.class);
c.carStarts();
}
}
1883 × 846px
100%
ENG

```java
public class Engine
{
	private String type;

	public void setType(String type) {
		this.type = type;
	}

	public void engineWorking()
	{
		System.out.println(type+" Engine starts working");
	}
}
```

```java
public class Car
{
	private String model;

	@Autowired
	private Engine engine;

	public void setModel(String model) {
		this.model = model;
	}

	public void carStarts()
	{
		engine.engineWorking();
		System.out.println("My car i.e '"+model+"' starts");
	}
}
```

```java
@Configuration
public class SpringConfigFile
{
	@Bean
	public Engine engine()
	{
		Engine engine = new Engine();
		engine.setType("V6");
		return engine;
	}

	@Bean
	public Car car()
	{
		Car c = new Car();
		c.setModel("Tata Nexon");
		return c;
	}
}
```




```java
public class Main
{
	public static void main(String[] args)
	{
		ApplicationContext context = new AnnotationConfigApplicationContext(SpringConfigFile.class);

		Car c = context.getBean(Car.class);
		c.carStarts();
	}
}
```

```
=> Java + Annotation Based Autowiring :-
    -> In this case we have to use @Autowired annotation
    -> @Autowired annotation can be used with field (property), setter method or constructor

    -> If there are 2 bean objects which are ready to be injected in the bean then there will be
    confusion. To remove this confusion we can use one annotation i.e. @Qualifier

    -> Note : We can use @Autowired annotation in case of XML spring configuration file
```

[

Field error in object 'target' on field 'name': rejected value [null];
codes [key_name.target.name,key_name.name,key_name.java.lang.String,key_name];
arguments [];
default message [Name is not valid],

Field error in object 'target' on field 'rollno': rejected value [0];
codes [key_rollno.target.rollno,key_rollno.rollno,key_rollno.int,key_rollno];
arguments [];
default message [Rollno cannot be 0],

Field error in object 'target' on field 'phoneno': rejected value [628383];
codes [key_phoneno.target.phoneno,key_phoneno.phoneno,key_phoneno.java.lang.String,key_phoneno];
arguments [];
default message [Phone no is not valid]

]

```
    List<ObjectError> list = dataBinder.getBindingResult().getAllErrors();

    if(list.isEmpty())
    {
        std.display();
    }
    else
    {
        for(ObjectError oe : list)
        {
            System.err.println(oe.getDefaultMessage());
        }
    }
}
```

```
=> Bean Validations :-
    -> Bean validation is the way by which we can check the proper data in the bean object
    -> We can achieve bean validations by multiple ways which are as follows :-
        1. Using "Validator" interface
        2. Using JSR-303 Bean Validation
        3. Using "@Valid" and "@Validate" annotations
        4. Using annotations of SpEL (Spring Expression Language)
           etc
```

==========

```
=> "Validator" interface :-
    -> Syntax :-
        interface Validator
        {
            public boolean supports(Class clazz)

            public void validate(Object obj, Errors errors)
        }
```

==========

```
=> DataBinder :-
    -> It is a class in sprnig used for data binding
    -> It is responsible for binding the data from HTTP request parameters to java objects

    -> NOTE :
        = It is not responsible for validation itself, but it can be configured with validators to
        perform validation on the bound data
        = The validation results are stored in a separate object of the type "BindingResult" which can be
        accessed after the data bind process
```

```
=> BindingResult :-
    -> It is an interface that represents the result of data binding and validations
    -> Implemented class of BindingResult interface are :-
        1. MapBindingResult
        2. BeanPropertyBindingResult
```

```
<bean class="in.sp.validators.StudentValidator" id="stdValId">
    <property name="resource" value="/in/sp/resources/error_messages.properties" />
</bean>
```

=> Logging :-

-> Logging is the process of tracking or recording important information, events, messages or issues that occur during the execution of our application.

-> The log files generated during logging process will help the developers or system administrators to monitor the application behaviour, diagous issues and track the errors

-> Use of logging :-

1. Error Tracking and Debugging :-
   = In web applications we can track the errors like error generated during form submission
2. Security Monitering :-
   = We can track security-related events such as failed to login attempts or unautorized acces attempts
3. Auditing & Compliance :-
   = A financial application might log all the financial transactions including the details of transaction, its time, location etc
4. Performace Analysis :-
   = We can track the time taken by our application to perform any event or respond
5. System Health Monitoring :-
   = In server enviornment we can track the memory usage, CPU load and other metrics
6. Deployment & Release Management :-
   = We can track the version number, time of releasing the application etc

etc

-> Where we can use logging :-

1. Software Development
   = Web Development
   = Mobile App Development
2. DevOps & Infrastructure :-
   = Server Applications
   = Databases
3. Networking & Security :-
   = Firewall & Security Appliances
   = Network Servers
4. Cloud Computing :-
   = Cloud Servers
   = Serverless Computing
5. Industrial Automation & IoT :-
   = Industrial Control Systems
   = IoT Devices

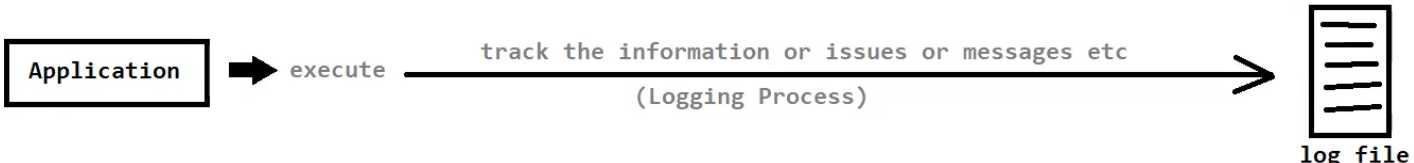
etc

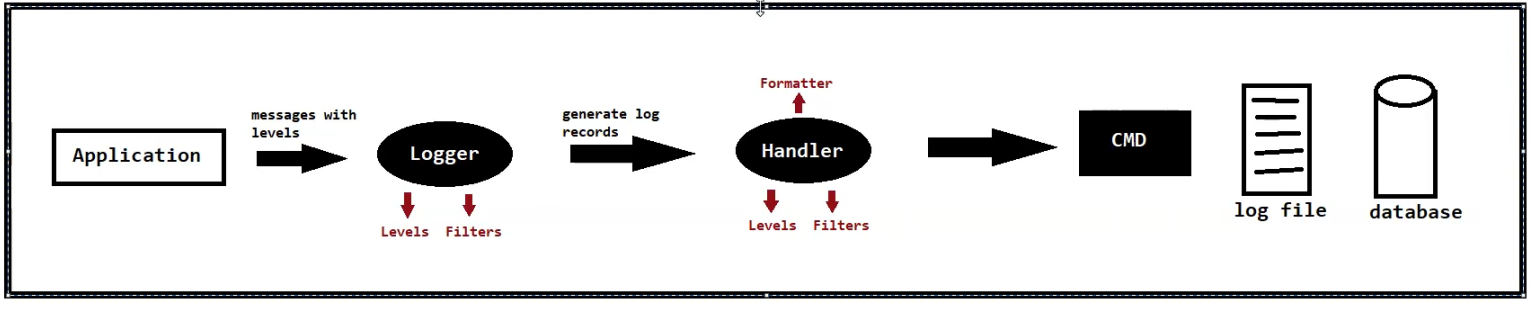
-> Logging process is supported by many languages i.e. java, python, php, JavaScript, Node JS etc

-> Logging was introduced in JDK 1.4 version
-> There are a lot of API's & Frameworks for logging :-

1. Java Logging API
2. Log4j
3. Logback
4. Tinylog

5. SLF4j
6. JCL (Jakarta Commons Logging) - old name was Apache Commons Logging
   (5th and 6th are logging wrappers)


Application
execute
track the information or issues or messages etc
(Logging Process)
log file


Application
messages with levels
Logger
Levels
Filters
generate log records
Formatter
Handler
Levels
Filters
CMD
log file
database

```
hello 1
Aug 05, 2023 6:32:12 PM in.sp.logging.Test1 main
SEVERE: Server is not responding
Aug 05, 2023 6:32:12 PM in.sp.logging.Test1 main
WARNING: 3 invalid login attempts
Aug 05, 2023 6:32:12 PM in.sp.logging.Test1 main
INFO: this is information message
hello 2
```

```
main method starts
m1 method starts
Aug 05, 2023 6:36:48 PM in.sp.logging.Test2 m1
SEVERE: m1 method severe message
Aug 05, 2023 6:36:48 PM in.sp.logging.Test2 m1
WARNING: m1 method warning message
Aug 05, 2023 6:36:48 PM in.sp.logging.Test2 m1
INFO: m1 method info message
m1 method ends
Aug 05, 2023 6:36:48 PM in.sp.logging.Test2 main
SEVERE: main method severe message
Aug 05, 2023 6:36:48 PM in.sp.logging.Test2 main
WARNING: main method warning message
Aug 05, 2023 6:36:48 PM in.sp.logging.Test2 main
INFO: main method info message
main method ends
```

```java
try
{
    System.out.println("hello 1");

    FileHandler fileHandler = new FileHandler("D:\\mydemo.log");
    SimpleFormatter simpleFormatter = new SimpleFormatter();
    fileHandler.setFormatter(simpleFormatter);

    Logger logger = Logger.getLogger("Test1");
    logger.addHandler(fileHandler);

    logger.log(Level.SEVERE, "Server is not responding");
    logger.log(Level.WARNING, "3 invalid login attempts");
    logger.log(Level.INFO, "this is information message");

    System.out.println("hello 2");
}
catch(Exception e)
{
    e.printStackTrace();
}
```

mydemo - Notepad

File Edit Format View Help

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE log SYSTEM "logger.dtd">
<log>
<record>
  <date>2023-08-05T13:10:21.279306100Z</date>
  <millis>1691241021279</millis>
  <nanos>306100</nanos>
  <sequence>0</sequence>
  <logger>Test1</logger>
  <level>SEVERE</level>
  <class>in.sp.logging.Test3</class>
  <method>main</method>
  <thread>1</thread>
  <message>Server is not responding</message>
</record>
<record>
  <date>2023-08-05T13:10:21.440867300Z</date>
  <millis>1691241021440</millis>
  <nanos>867300</nanos>
  <sequence>1</sequence>
```

mydemo - Notepad

File Edit Format View Help

Aug 05, 2023 6:41:48 PM in.sp.logging.Test3 main
SEVERE: Server is not responding
Aug 05, 2023 6:41:48 PM in.sp.logging.Test3 main
WARNING: 3 invalid login attempts
Aug 05, 2023 6:41:48 PM in.sp.logging.Test3 main
INFO: this is information message

```
Enter no1
10
Aug 05, 2023 6:50:50 PM in.sp.calc.Calculator1 main
INFO: user has provided no1 i.e. 10
Enter no2
20
Aug 05, 2023 6:50:52 PM in.sp.calc.Calculator1 main
INFO: user has provided no2 i.e. 20
Select Any One Symbol (+, -, *, /)
+
Aug 05, 2023 6:50:54 PM in.sp.calc.Calculator1 main
INFO: user has provided symbol i.e. +
Sum is : 30
Aug 05, 2023 6:50:54 PM in.sp.calc.Calculator1 main
INFO: result is 30
```

```
Logger logger = Logger.getLogger("Calculator1");

Scanner scanner = new Scanner(System.in);

System.out.println("Enter no1");
int no1 = scanner.nextInt();
logger.log(Level.INFO, "user has provided no1 i.e. "+no1);

System.out.println("Enter no2");
int no2 = scanner.nextInt();
logger.log(Level.INFO, "user has provided no2 i.e. "+no2);

System.out.println("Select Any One Symbol (+, -, *, /)");
String symbol = scanner.next();
logger.log(Level.INFO, "user has provided symbol i.e. "+symbol);

int res;
```

```
Logger.setLevel(Level.SEVERE);
```

```
1. Java Logging API :-
    -> Java Logging API was introduced in JDK 1.4 version
    -> It is pre-defined in JDK so no need to download any jar file or provide dependency
    -> This API is present in "java.util.logging" package
    -> Components of logging :-
        = There are 2 components in logging :-
            i. Logger
            ii. Handler

-----------------------------------------
```

---

i. Logger :-

-> Logger is an object in logging framework that you use to emit the log messages

-> Some levels of Logger are :-

1. SEVERE (highest value)
2. WARNING
3. INFO
4. CONFIG
5. FINE
6. FINER
7. FINEST (lowest value)

ii. Handler :-

-> Handler is also an object that listens the messages at or above a specified minimum security level

-> Handler will take the messages and post it to the provided medium like console or file or database

-> There are 5 handlers :-

1. ConsoleHandler
2. FileHandler
3. StreamHandler
4. SocketHandler
5. MemoryHandler

=> Log4j API :-
    -> Log4j is an open-source logging API for java
    -> Log4j was introduced in 2001
    -> It is used to store the log details and make it available for tracking the errors or messages or instructions etc
    -> Log4j is fast, reliable and flexible logging API

    -> Components of Log4j :-
        i. Logger
        ii. Appender

---

i. Logger :-
    -> It is an object or component which generates the log messages
    -> Security levels of Logger are :-
        1. OFF (highest value)
        2. Fatal - fatel()
        3. ERROR - error()
        4. WARN - warn()
        5. INFO - info()
        6. DEBUG - debug()
        7. TRACE (lowest value) - trace()

ii. Appender :-

-> It is an object or component which determines where the log messages are sent for storage or display

-> Some Appenders in Log4j are :-

1. ConsoleAppender
2. FileAppender
3. WriterAppender
4. JDBCAppender
5. SocketAppender
6. TelnetAppender
7. SMPTAppender
8. SystlogAppender

-> NOTE : To use Log4j, we have to download one jar file i.e. log4j.*.jar

```
log4j.rootLogger = INFO, CA

log4j.appender.CA = org.apache.log4j.ConsoleAppender
log4j.appender.CA.layout = org.apache.log4j.PatternLayout
log4j.appender.CA.layout.ConversionPattern = %p %d %m%n
```

```
{
    System.out.println("--------- App Starts ---------");

    PropertyConfigurator.configure(System.getProperty("user.dir")+"/src/in/sp/resources/lo

    Logger logger = Logger.getLogger("Main");

    logger.fatal("This is fatal message");
    logger.error("This is error message");
    logger.warn("This is warn message");
    logger.info("This is info message");

    System.out.println("--------- App Stops ---------");
}
```

```
#configurations for FileAppender
log4j.appender.FA = org.apache.log4j.FileAppender
log4j.appender.FA.File = d:/log4jfile.log
log4j.appender.FA.layout = org.apache.log4j.PatternLayout
log4j.appender.FA.layout.ConversionPattern = [%p] - %d{dd/MM/yyyy hh:mm:ss aa} - %m%n
```

```
#configurations for file
log4j.appender.FA = org.apache.log4j.FileAppender
log4j.appender.FA.File = d:/log4jfile.log
log4j.appender.FA.layout = org.apache.log4j.PatternLayout
log4j.appender.FA.layout.ConversionPattern = [%p] - %d{dd/MM/yyyy hh:mm:ss aa} - %m%n

#configurations for console
log4j.appender.CA = org.apache.log4j.ConsoleAppender
log4j.appender.CA.layout = org.apache.log4j.PatternLayout
log4j.appender.CA.layout.ConversionPattern = %p %d %m%n
```

```java
package in.sp.main;

import org.apache.log4j.PropertyConfigurator;

public class Main
{
    public static void main(String[] args)
    {
        System.out.println("--------- App Starts ---------");

        PropertyConfigurator.configure(System.getProperty("user.dir")+"/src/in/sp/resources/l

        Logger logger = Logger.getLogger("Main");

        logger.fatal("This is fatal message");
        logger.error("This is error message");
        logger.warn("This is warn message");
        logger.info("This is info message");

        System.out.println("--------- App Stops ---------");
    }
}
```

Main.java | log4j.properties

```
log4j.rootLogger = INFO, FA

log4j.appender.FA = org.apache.log4j.FileAppender
log4j.appender.FA.File = d:/log4jfile.html
log4j.appender.FA.layout = org.apache.log4j.HTMLLayout
log4j.appender.FA.layout.Title = My Log Messages
```

=> Whenever we create an application, we always want to execute it for different countries having different languages and time zones and cultures

=> For this purpose we use "Internationalization" and "Localization"

=> Some data or information that is different for different countries :-

- = Number
- = Currency
- = Date
- = Time
- = Messages
- = Phone Numbers
- = Address
- etc

```
=> Internationalization :-
    -> It is also known as "I18N"
    -> Internationalization means designing and creating the application (web or software) in such a way
    that makes them easy to adapt for the people of different countries or cultures
    -> It's like making the foundation that can later be customized to fit for the preferences and
    languages of different countries


=> Localization
    -> It is also known as "L10N"
    -> It is like taking the foundation from Internationalization and customzing it for a specific reason
    or language or culture


=> To achieve Internationalization and Localization, java has provided some pre-defined classes :-
    1. Locale
    2. NumberFormat
       DecimalFormat
    3. DateFormat
       SimpleDateFormat
    4. ResourceBundle
       etc
```

```java
Locale locale = Locale.getDefault();

System.out.println("Default locale : "+locale);
System.out.println("Country Name : "+locale.getDisplayCountry());
System.out.println("Country Code : "+locale.getCountry());
System.out.println("Country Langauge : "+locale.getDisplayLanguage());
System.out.println("Language Code : "+locale.getLanguage());

System.out.println("-----------------------------------------------------");

Locale locale2 = new Locale("fr", "FR");
System.out.println("Provided locale : "+locale2);
System.out.println("Country Name : "+locale2.getDisplayCountry());
System.out.println("Country Code : "+locale2.getCountry());
System.out.println("Country Langauge : "+locale2.getDisplayLanguage());
System.out.println("Language Code : "+locale2.getLanguage());
```

```
String[] countryCode_arr = Locale.getISOCountries();
for(String countryCode : countryCode_arr)
{
    System.out.println(countryCode);
}
```

```
String[] countryCode_arr = Locale.getISOCountries();
for(String countryCode : countryCode_arr)
{
    Locale locale = new Locale("", countryCode);
    System.out.println(countryCode+" -> "+locale.getDisplayCountry());
}
```

```
String[] langaugeCode_arr = Locale.getISOLanguages();
for(String languageCode : langaugeCode_arr)
{
    Locale locale = new Locale(languageCode);
    System.out.println(languageCode+" -> "+locale.getDisplayLanguage());
}
```

```java
//Locale mylocale = Locale.JAPANESE;
//Locale mylocale = Locale.CHINESE;
Locale mylocale = new Locale("hi");

String[] countryCode_Arr = Locale.getISOCountries();
for(String countryCode : countryCode_Arr)
{
    Locale locale = new Locale("", countryCode);
    System.out.println(countryCode+" -> "+locale.getDisplayCountry()+" - "+locale.getDisplayCountry(myl
}
```

```java
int no1 = 123456789;
double no2 = 98765.43210;

System.out.println("--------------Below is Indian Format-----------------");
Locale locale1 = Locale.getDefault();
NumberFormat nf1 = NumberFormat.getInstance(locale1);
System.out.println(nf1.format(no1));
System.out.println(nf1.format(no2));


System.out.println("--------------Below is France Format-----------------");
Locale locale2 = new Locale("fr", "FR");
NumberFormat nf2 = NumberFormat.getInstance(locale2);
System.out.println(nf2.format(no1));
System.out.println(nf2.format(no2));
```

```
Locale locale1 = Locale.getDefault();
Currency currency = Currency.getInstance(locale1);
System.out.println(currency.getSymbol()+" -> "+currency.getDisplayName());
```

```java
int no1 = 123456789;
double no2 = 98765.43210;

Locale locale1 = Locale.getDefault();

NumberFormat nf1 = NumberFormat.getCurrencyInstance(locale1);
System.out.println(nf1.format(no1));
System.out.println(nf1.format(no2));
```

```java
int no1 = 123456789;
double no2 = 9876.543210;

//String pattern = "######.#####";
//String pattern = "######.##";
String pattern = "##,##,##.#####";

DecimalFormat dm1 = new DecimalFormat(pattern);
System.out.println(dm1.format(no1));
System.out.println(dm1.format(no2));
```

```java
Date date = new Date();

//System.out.println(date);

System.out.println("----------Below is Indian Format----------");
Locale locale1 = new Locale("en", "IN");
DateFormat df1 = DateFormat.getDateInstance(0, locale1);
System.out.println(df1.format(date));
```

```java
Date date = new Date();

//System.out.println(date);

//String pattern = "dd/MM/yyyy";
//String pattern = "dd/MMM/yyyy";
//String pattern = "dd/MMM/yy";
//String pattern = "dd MM yyyy";
//String pattern = "dd MMM yyyy";
//String pattern = "dd-MM-yyyy";
String pattern = "dd-MMM-yyyy HH:mm:ss";

SimpleDateFormat sdf1 = new SimpleDateFormat(pattern);
System.out.println(sdf1.format(date));
```

```
Choose any one option from below :-
1. English - US
2. Hindi - India
---------------------------------------
1
```

```
LoginFrame()
{
    JFrame jf = new JFrame();
    jf.setSize(600, 600);
    jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    jf.setLayout(null);

    JLabel jl_login_title = new JLabel("Login Here");
    jl_login_title.setBounds(250, 50, 100, 30);
    jf.add(jl_login_title);

    JLabel jl_email_title = new JLabel("Enter Email");
    jl_email_title.setBounds(60, 150, 100, 30);
    jf.add(jl_email_title);

    JTextField jt_email = new JTextField();
    jt_email.setBounds(60, 200, 300, 40);
    jf.add(jt_email);

    jf.setVisible(true);
```

```
    JTextField jt_email = new JTextField();
    jt_email.setBounds(100, 180, 300, 40);
    jf.add(jt_email);

    JLabel jl_pass_title = new JLabel("Enter Passsword :");
    jl_pass_title.setBounds(100, 270, 130, 30);
    jf.add(jl_pass_title);

    JTextField jt_pass = new JTextField();
    jt_pass.setBounds(100, 300, 300, 40);
    jf.add(jt_pass);

    JButton jb_login = new JButton("Login");
    jb_login.setBounds(200, 400, 100, 40);
    jf.add(jb_login);

    jf.setVisible(true);
}
public static void main(String[] args)
{
```

=> ResourceBundle :-
    -> It is an abstract class which is present in "java.util" package
    -> It is used to achieve internationalization wrt messages (string) or images etc

    -> NOTE : We have to use properties file

=> Naming convention of "properties file" :-

    -> basename_languageCode_countryCode_systemVarient.properties

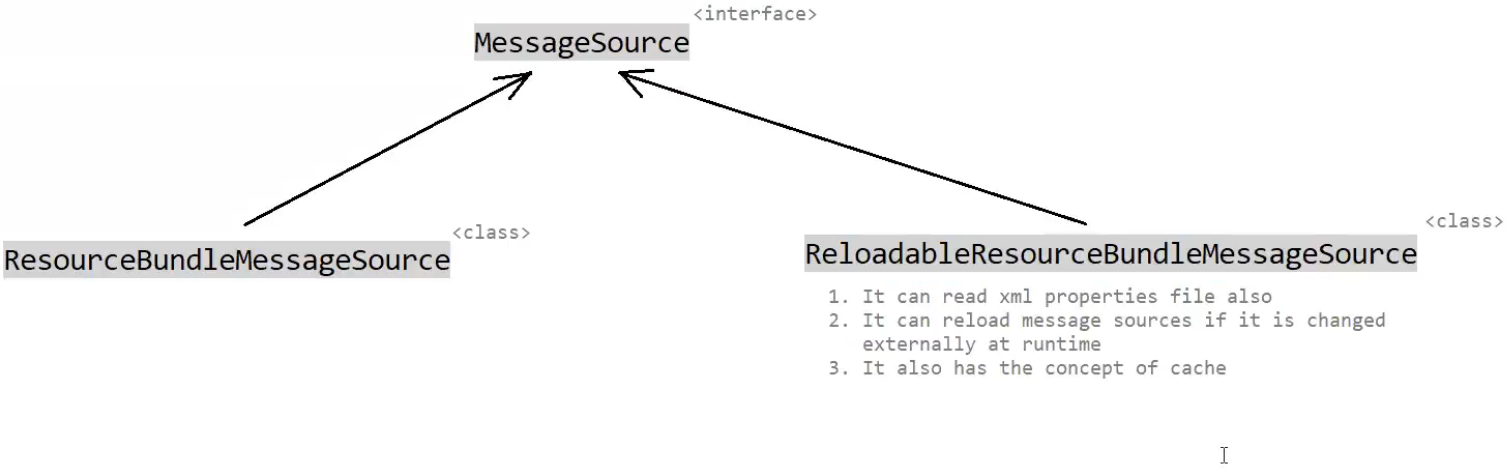
    -> For example :-
        MessageBundle_en_US.properties
        MessageBundle_hi_IN.properties

==========

```xml
<!-- bean definitions here -->
<bean class = "org.springframework.context.support.ResourceBundleMessageSource"
            id = "mgS">
            <!--ye inbuilt method hai usme set kar rahe hai value  -->
        <property name = "basename" value = "me/resources/MessageBundle"/>
</bean>

<bean class = "me.beans.Student" id= "std">
<!-- then uska reference hum yaha set kar rahe hai-->
    <property name ="messageSource" ref ="mgS"/>
</bean>
```

```java
blic class Student {
  private MessageSource messageSource;

  public void setMessageSource(MessageSource messageSource) {
      this.messageSource = messageSource;
  }

  public  void display() {
      Locale locale = new Locale("en","US");
      String message = messageSource.getMessage("key_message",null, locale);
      System.out.println(message);
  }
```

```java
@Bean
public ResourceBundleMessageSource rsmSrcId()
{
    ResourceBundleMessageSource obj = new ResourceBundleMessageSource();
    obj.setBasename("in/sp/resources/MessageBundle");

    return obj;
}

@Bean
public Student stdId()
{
    Student std = new Student();
    std.setMsgsource(rsmSrcId());

    return std;
}
```


MessageSource
<interface>
ResourceBundleMessageSource
<class>
ReloadableResourceBundleMessageSource
<class>
1. It can read xml properties file also
2. It can reload message sources if it is changed externally at runtime
3. It also has the concept of cache

```
=> SpEL :-
    -> SpEL stands for "Spring Expression Language"
    -> SpEL is an "Expression Langauge" which allows you to define and evaluate the expressions at runtime
    -> Features supported by SpEL are :-
        1. Operators :-
            = Arithmetic Operators
                -> +, -, *, / etc
            = Relational Operators
                -> ==   or  eq
                -> !=   or  ne
                -> >    or  gt
                -> <    or  lt
                -> >=   or  ge
                -> <=   or  le
            = Logical Operators
                -> &&   or  and
                -> ||   or  or
                -> !    or  not
            = Ternary Operator
                -> variable = conditional-expression ? expression1 : expression2
            = Type Operator
                -> T(ClassName)
```

```
    2. Expressions :-
        = Literal Expressions
        = Method Invocation
        = Constructor Invocation
        = Regular Expressions (RegEx)
        = Class Expressions
        = Templated Expressions
          etc
    3. Accessing arrays, lists, maps etc
    4. Bean References
       etc
-> How to use SpEL :-
    = We can use SpEL by 2 ways :-
        1. Using pre-defined interfaces and classes
        2. #{expression}
```

==========

=> Pre-defined interfaces and classes in SpEL are :-

1. ExpressionParser <interface>
2. SpelExpressionParser <class>

3. Expression <interface>
4. SpelExpression <class>

5. EvaluationContext <interface>
6. StandardEvaluationContext <class>

-> NOTE : Above all interfaces and classes are present in "org.springframework.expression"

1. ExpressionParser :-
    -> It is an interface which is responsible to parse (resolve) a string expression
    -> Method :-
        = parseExpression(-)

2. SpelExpressionParser :-
    -> It is an implemented class for ExpressionParser

---

3. Expression :-
    -> It is an interface which is responsible to evaluate the string expression
    -> Methods :-
        = getValue(-)
        = getValueType()
        = getValueTypeDescriptor()
        = getExpressionString()
          etc

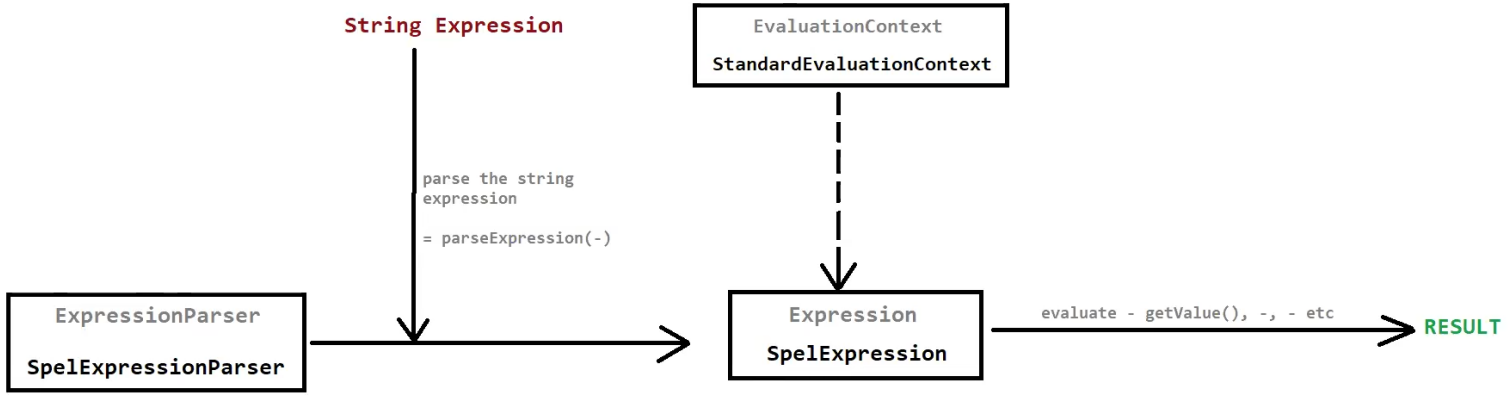
4. SpelExpression :-
    -> It is an implemented class for Expression

```
ExpressionParser parsor = new SpelExpressionParser();

Expression expression = parsor.parseExpression(" 10+20 ");
Object obj =  expression.getValue();
System.out.println(obj);
```

```
ExpressionParser parsor = new SpelExpressionParser();

Expression expression = parsor.parseExpression(" '10+20' ");
Object obj =  expression.getValue();
System.out.println(obj);
```

```
ExpressionParser parsor = new SpelExpressionParser();

Expression expression = parsor.parseExpression(" 10+20 ");
Object obj =  expression.getValue();
System.out.println(obj);


//-----------------------------------------------------------------

ExpressionParser parsor = new SpelExpressionParser();

EvaluationContext context = new StandardEvaluationContext();
context.setVariable("no1", 100);
context.setVariable("no2", 200);

Expression expression = parsor.parseExpression(" #no1 + #no2");
Object obj = expression.getValue(context);
System.out.println(obj);
```

SpEL Workflow



---

3. Expression :-
   -> It is an interface which is responsible to evaluate the string expression
   -> Methods :-
      = getValue(-)
      = getValueType()
      = getValueTypeDescriptor()
      = getExpressionString()
        etc

4. SpelExpression :-
   -> It is an implemented class for Expression

---

5. EvaluationContext :-
   -> It is an interface which is responsible for providing the context in which expressions are evaluated. It holds the variables, functions and other contextual information required for evaluating SpEL expression

6. StandardEvaluationContext :-
   -> It is an implemented class for EvaluationContext

==========

```java
ExpressionParser parsor = new SpelExpressionParser();

Expression expression = parsor.parseExpression(" 10 == 20 ");
System.out.println(expression.getValue());


Expression expression = parsor.parseExpression(" 10 ne 20 ");
System.out.println(expression.getValue());


Expression expression = parsor.parseExpression(" (10 < 20) && (30 < 40) ");
System.out.println(expression.getValue());


Expression expression = parsor.parseExpression(" (10 > 20) ? '111' : '222' ");
System.out.println(expression.getValue());
```

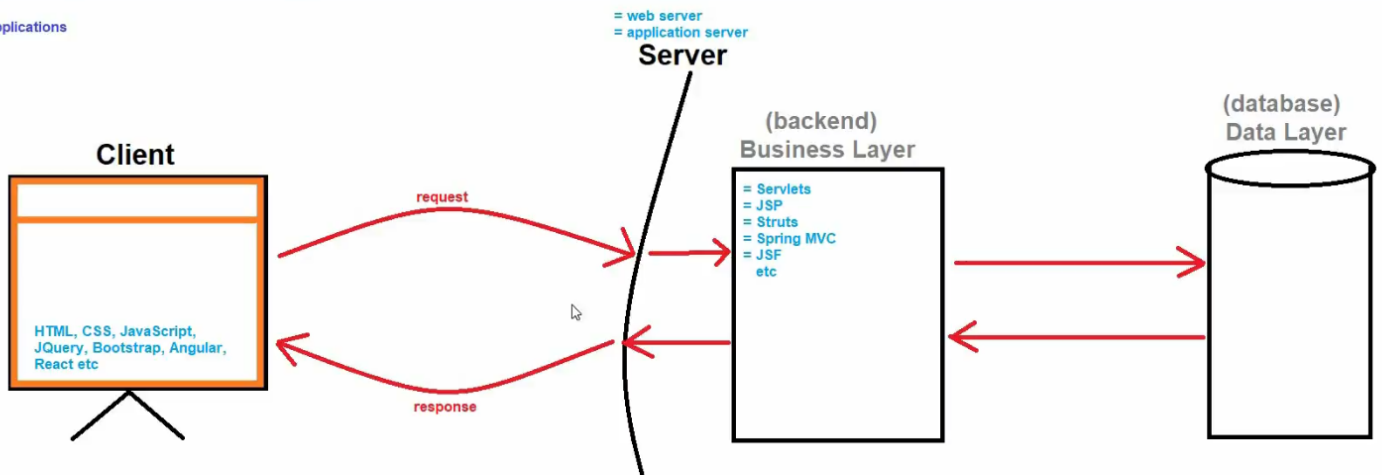
```
ExpressionParser parsor = new SpelExpressionParser();

String str_exp = " 'deepak panwar'.length() ";
Expression expression = parsor.parseExpression(str_exp);
System.out.println(expression.getValue());


String str_exp = " 'deepak panwar'.toUpperCase() ";
Expression expression = parsor.parseExpression(str_exp);
System.out.println(expression.getValue());
```

```
ExpressionParser parsor = new SpelExpressionParser();


String str_exp = " 'deepak' matches 'Deepak'  ";
Expression expression = parsor.parseExpression(str_exp);
System.out.println(expression.getValue());


String str_exp = " 'deepak' matches '[a-zA-Z]{5,15}' ";
Expression expression = parsor.parseExpression(str_exp);
System.out.println(expression.getValue());
```

```
ExpressionParser parsor = new SpelExpressionParser();

Expression expression = parsor.parseExpression(" T(in.sp.main1.MyClass).m1() ");
expression.getValue();


Expression expression = parsor.parseExpression(" T(in.sp.main1.MyClass).m3() ");
Object obj = expression.getValue();
System.out.println(obj);


Expression expression = parsor.parseExpression(" T(in.sp.main1.MyClass).m3() ");
int i = (int) expression.getValue();
System.out.println(i);


Expression expression = parsor.parseExpression(" T(in.sp.main1.MyClass).m3() ");
int i = expression.getValue(Integer.class);
System.out.println(i);
```

```
Expression expression = parsor.parseExpression(" new in.sp.main1.MyClass().m2() ");
expression.getValue();
```

Architecture of Web Applications



=> Editions In Java :-

-> There are 3 editions in java :-

1. J2SE (Java 2 Standard Edition) - Core Java
2. J2EE (Java 2 Enterprise Edition) - Advance Java
3. J2ME (Java 2 Micro Edition) - Mobiles, Embded System (remotes, ATM's, TV, Washing Machines etc)

=> Types of applications in java :-

-> We can create 2 types of applications in java :-

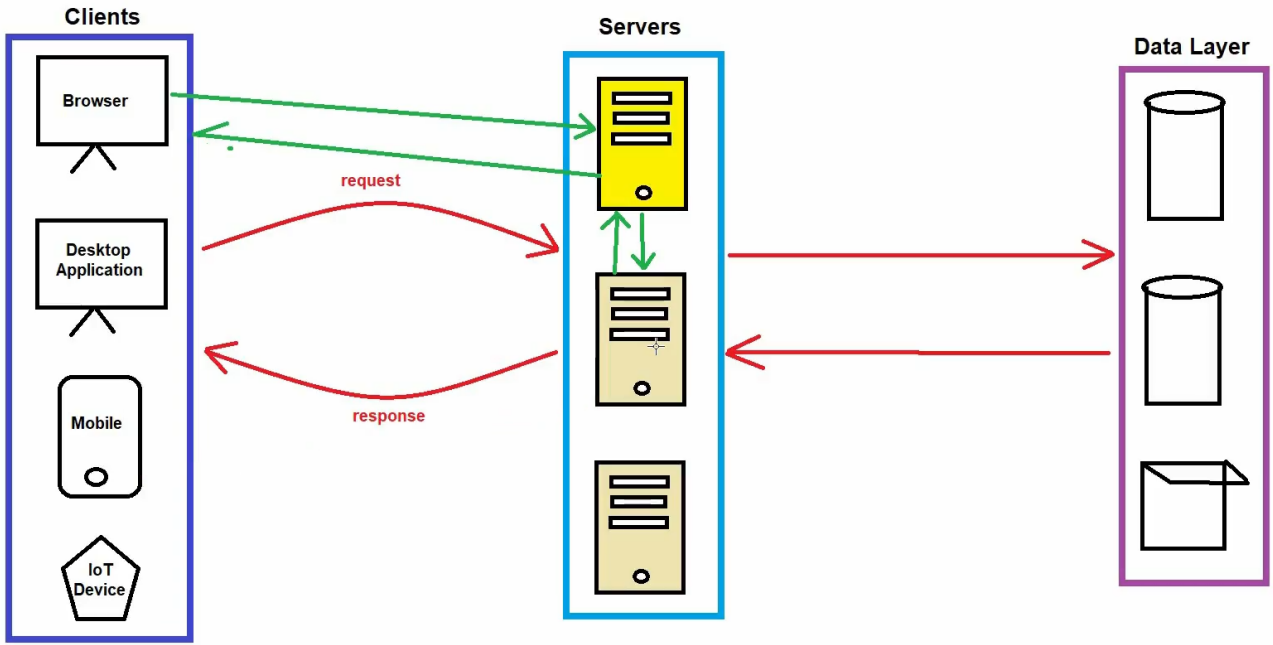
1. Standalone Applications
   - = These are the applications which are executed only on single system
   - = These applications can be developed using J2SE
   - = These applications does not follows the client-server architecture
   - = There are 2 types of standalone applications :-
     a. CUI (Character User Interface) Applications
        - Console Based App or Command Line User Interface or Text-Based App
     b. GUI (Graphical User Interface) Applications
2. Enterprise Applications

=> What is Enterprise ?

-> "Enterprise" term is used for large scale companies which has multiple departments, levels, divisons or groups

-> For eg :-

- = TATA Group : Consumer and retail, Hotels, IT, Automobiles, Steel, Power etc
- = Mahindra Group : IT, Automobiles, Defence, Education, Financial Services etc etc



=> What is enterprise applications ?

-> "Enterprise Applications" are large-scale, distributed, transaction and highly available applications which are designed to support the enterprise business requirements

-> To develop enterprise applications we have to use a lot of technologies, multiple design patterns, system architectures .

=> Web Applications :-

-> Client : Browser

-> Server : Web Server or Application Server

-> Technologies Used : Servlet, JSP, Spring MVC, JSF, Play Framework, Struts etc

-> Architecture :-

=> Distributed Applications :-

-> Client : Browser, Desktop Application, Mobile Application, IoT Device etc

-> Server : Application Server

-> Technologies Used : EJB (Enterprise Java Beans), Spring framework, JPA (Java Persistence API), Hibernate, JTA (Java Transaction API), JMS (Java Message Service) etc

-> Architecture :-

=> Difference between Web Server and Application Server :-

= Web Server is lightweight
  Application Server is heavy weight

= Web server contains only web containers (servlet container, JSP container etc)
  Application server contains web container + EJB container

= Web server is good for static contents like html pages
  Application server is good for dynamic contents

= Web server consumes less resources i.e. CPU, memory etc
  Application server use more resources

= Web server examples are : Apache Tomcat, Resin etc
  Application server examples are : Weblogic, JBoss, Websphere etc

=> What is framework ?

-> Frameworks are the pre-written code acting as a template which can be used or customized by the developers

-> In simple terms we can say that frameworks are the collections of API's and tools which can be used to develop projects

-> For example : Spring, Struts, Hibernate, Angular, React etc

=> Advantages of frameworks :-

-> Fast development speed

-> Less code (because it removes the boilerplate code)

-> Support API integration

-> Customizable (open source)

-> Easy to debug

-> Good documentation support

etc

=> Types of frameworks :-

-> There are 2 types of frameworks :-

1. Web Framework
   = eg. Struts, JSF etc
2. Application Framework
   = eg. Spring etc