Java EE - Day 1

From browser application:

http://localhost:8081/demo-web-app/

Received by Server running locally and listening at port: 8081 ie. Tomcat

Now Tomcat verifies whether demo-web-app application is deployed ? YES

Next Step, any pattern after demo-web-app/ ? NO

Next Step: Go to web.xml, check welcome file list, if any the file names
Specified in web.xml, is it present under web app folder ? YES

Now Tomcat will serve that file to the client/browser

Note: Project name is also called as context root

Static Pages or Dynamic Pages ?

home.html and default.html are static pages

How to create dynamic pages:

We require Scripting languages to generate content dynamically:

1. Java. 2. Python.

The two widely used technologies for developing dynamic web pages are:
1.Servlets
2.JSP –Java Server Pages

***A* Servlet** is a specialized Java class that runs on a web server to generate dynamic web pages.
Servlets work on a request -response programming model i.e. they accept requests from the clients , generate responses dynamically and send the responses in a format such as ***html*** to the clients/browsers.

```
package com.wipro.controller;
import java.io.IOException;
import java.io.PrintWriter;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
/*
* http://localhost:8081/dynamic-web-app-demo/hello
*
* Since default HTTP method is GET, control enters into gdoGet() method
*
*    HttpServlet -> GenericServlet -----> Servlet
*
*    Servlet is an interface, GenericServlet is a class that implements Servlet,
*    HttpServlet is class that extends GenericServlet.
*
*
*    GenericServlet can handle any type of protocol( Http, ftp, SMTP etc) whereas
HttpServlet
*    can handle only Http protocol. Since most of the web applications are http-base, we
create our own
*    servlet that extend HttpServlet.
*
*/
@WebServlet("/hello")
```

```
public class HelloWorldServlet extends HttpServlet {
	private static final long serialVersionUID = 1L;

	protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

		PrintWriter out = response.getWriter();
		//dynamically generate the web page, writes into response object
		out.println("<html><body><h1><font color='red'> Welcome to My Dynamic
Page</font></h1></body></html>");
	}

	protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

		doGet(request, response);
	}
}
```

—----------

```
<servlet>
  <description></description>
  <display-name>HelloWorldServlet</display-name>
  <servlet-name>HelloWorldServlet</servlet-name>
  <servlet-class>com.wipro.controller.HelloWorldServlet</servlet-class>
 </servlet>
 <servlet-mapping>
  <servlet-name>HelloWorldServlet</servlet-name>
  <url-pattern>/hello</url-pattern>
 </servlet-mapping>
```

—----------

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
```

```
		<h2>Login Form</h2>
		<form action="" method="get">
			Enter Userid: <input   type="text" name="userid" size="20" />
			Enter Password: <input type="text" name="password"  size="20"/>
			<input type="submit"  value="Login"/>
		</form>
</body>
</html>
```

—----------

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
		<h2>Login Form</h2>
		<form action="login" method="post">
			Enter Userid: <input   type="text" name="userid" size="20" /> <br>
			Enter Password: <input type="password" name="password"
size="20"/><br>
			<input type="submit"  value="Login"/>
		</form>
</body>
</html>
```

—----------

```
package com.wipro.controller;
import java.io.IOException;
import java.io.PrintWriter;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
/*
```

```
* If Http method is GET, the form data is appended to the url and sent to the server in
the foll. format:
*
*                                   url?querystring
*
*                   query string format:
*                                   name=value&name=value
*
*          Ex.
*
http://localhost:8081/dynamic-web-app-demo/login?userid=Srini&password=Srini%4012
3
*
*                   While sending sensitive data or large amount of data to the server, apply
HttP POST/PUT methods
*/
@WebServlet("/login")
public class LoginServlet extends HttpServlet {
        private static final long serialVersionUID = 1L;


        protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
                try {
                        PrintWriter out = response.getWriter();
                        String userid = request.getParameter("userid");
                        String password =  request.getParameter("password");

                        System.out.println(userid+","+password);

                        out.println("<html><body><h2>Your Credentials:"+
userid+","+password+"</h2></body></html>");


                }catch(Exception e) {

response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR,e.getMe
ssage());
                }
```

```
	}

	protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

		doGet(request, response);
	}
}
```

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
<!-- Font Awesome -->
<link
 href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0/css/all.min.css"
 rel="stylesheet"
/>
<!-- Google Fonts -->
<link
 href="https://fonts.googleapis.com/css?family=Roboto:300,400,500,700&display=swap"
 rel="stylesheet"
/>
<!-- MDB -->
<link
 href="https://cdnjs.cloudflare.com/ajax/libs/mdb-ui-kit/7.3.0/mdb.min.css"
 rel="stylesheet"
/>
<style>
	.intro {
 height: 100%;
}
@media (min-height: 300px) and (max-height: 450px) {
 .intro {
   height: auto;
 }
}
```

```
.gradient-custom {
 background: radial-gradient(50% 123.47% at 50% 50%, #00FF94 0%, #720059 100%),
linear-gradient(121.28deg, #669600 0%, #FF0000 100%), linear-gradient(360deg,
#0029FF 0%, #8FFF00 100%), radial-gradient(100% 164.72% at 100% 100%, #6100FF
0%, #00FF57 100%), radial-gradient(100% 148.07% at 0% 0%, #FFF500 0%, #51D500
100%);
 background-blend-mode: screen, color-dodge, overlay, difference, normal;
}
</style>
</head>
<body>
<section class="intro">
 <div class="mask d-flex align-items-center h-100" style="background-color:
#D6D6D6;">
   <div class="container">
    <div class="row justify-content-center">
     <div class="col-12 col-md-8 col-lg-6 col-xl-5">
      <div class="card" style="border-radius: 1rem;">
       <div class="card-body p-5 text-center">
        <div class="my-md-5 pb-5">
         <h1 class="fw-bold mb-0">Welcome</h1>
         <i class="fas fa-user-astronaut fa-3x my-5"></i>
                                <form action="login" method="post">
                    <div class="form-outline mb-4">
                     <input type="email" id="typeEmail" name="email" class="form-control
form-control-lg" />
                     <label class="form-label" for="typeEmail">Email</label>
                    </div>

                    <div class="form-outline mb-5">
                     <input type="password" id="typePassword" name="password"
class="form-control form-control-lg" />
                     <label class="form-label" for="typePassword">Password</label>
                    </div>

                    <button class="btn btn-primary btn-lg btn-rounded gradient-custom
text-body px-5" type="submit">Login</button>

                   </div>
                                </form>
```

```
            <div>
              <p class="mb-0">Don't have an account? <a href="#!" class="text-body
fw-bold">Sign Up</a></p>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
 </div>
</section>
</body>
</html>
```

```
package com.wipro.controller;
import java.io.IOException;
import java.io.PrintWriter;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
/*
* If Http method is GET, the form data is appended to the url and sent to the server in
the foll. format:
*
*                                   url?querystring
*
*                  query string format:
*                                   name=value&name=value
*
*          Ex.
*
http://localhost:8081/dynamic-web-app-demo/login?userid=Srini&password=Srini%4012
3
*
*                  While sending sensitive data or large amount of data to the server, apply
HttP POST/PUT methods
*/
```

```
@WebServlet("/login")
public class LoginServlet extends HttpServlet {
	private static final long serialVersionUID = 1L;


	protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
		try {
			PrintWriter out = response.getWriter();
			String email = request.getParameter("email");
			String password =  request.getParameter("password");

			System.out.println(email + "," +password);

			out.println("<html><body><h2>Your Credentials:"+
email+","+password+"</h2></body></html>");


		}catch(Exception e) {

response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR,e.getMe
ssage());
		}

	}

	protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

		doGet(request, response);
	}
}
```

—----------

ServletConfig and ServletContext objects.

For every Servlet object, Servlet Container implicitly create one ServletConfig object which contains initialization values provided to the Servlet..

When a first request for a Servlet comes from the client, Servlet container instantiates the Servlet class, instance/object of the servlet is created and at that time if we provide some initialization values, those values values will be placed in ServletConfig object.

When a second request comes for the same Servlet , will another instance be created? NO.

A new thread is spawned and that thread will enter into the servlet and execute doGet()/doPost() methods.

For all subsequent threads, separate threads are spawned and they execute the method(s) of the Servet.

Servlet Lifecycle methods:

init() : Executed only once during instantiation of the servlet.
service() : converted into doGet()/doPost() .. methods in HttpServlet class get executed for every request/thread.
destroy() : executed only once when the servlet is removed from servlet container

```
package com.wipro.controller;
import java.io.IOException;
import java.io.PrintWriter;
import jakarta.servlet.ServletConfig;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebInitParam;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
/*
* http://localhost:8081/dynamic-web-app-demo/hello
*
* Since default HTTP method is GET, control enters into gdoGet() method
*
*    HttpServlet -> GenericServlet -----> Servlet
*
*    Servlet is an interface, GenericServlet is a class that implements Servlet,
*    HttpServlet is class that extends GenericServlet.
```

```
*
*
*    GenericServlet can handle any type of protocol( Http, ftp, SMTP etc) whereas
HttpServlet
*    can handle only Http protocol. Since most of the web applications are http-base, we
create our own
*    servlet that extend HttpServlet.
*
*/
@WebServlet(urlPatterns= "/hello" ,
                        initParams = {
                                                @WebInitParam(name = "user", value =
"Srini"),
                                                @WebInitParam(name = "job", value =
"Trainer")
                                        }
                        )
public class HelloWorldServlet extends HttpServlet {
        private static final long serialVersionUID = 1L;

        protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

                PrintWriter out = response.getWriter();
                //dynamically generate the web page, writes into response object
                out.println("<html><body><h1><font color='red'> Welcome to My Dynamic
Page</font></h1></body></html>");

                //getting reference to ServletConfig object of HelloWorldServlet instance
                ServletConfig config = this.getServletConfig();
                //ServletConfig object contains initialization values of the servlet
                out.println(config.getInitParameter("user"));
                out.println(config.getInitParameter("job"));


        }

        protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
```

```
            doGet(request, response);
        }
}
```

ServletContext object:

There is only one ServletContext object per web application.

The data stored in ServletContext object is global in the sense all the web components the application can access the contents of ServletContext object.

SerlvetContext is created implicitly when the web application is deployed on to the web server and implicitly removed when the web application is undeployed from the container.

How to initialize a ServletContext object? **We can initialize only in web.xml file**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="https://jakarta.ee/xml/ns/jakartaee"
xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
https://jakarta.ee/xml/ns/jakartaee/web-app_6_0.xsd" id="WebApp_ID" version="6.0">
 <display-name>dynamic-web-app-demo</display-name>
 <welcome-file-list>
   <welcome-file>login.html</welcome-file>
   <welcome-file>index.jsp</welcome-file>
   <welcome-file>index.htm</welcome-file>
   <welcome-file>default.html</welcome-file>
   <welcome-file>default.jsp</welcome-file>
   <welcome-file>default.htm</welcome-file>
 </welcome-file-list>

 <context-param>
        <param-name>database</param-name>
```

```
        <param-value>Oracle</param-value>
</context-param>
</web-app>
```

```
ServletContext context = this.getServletContext();

out.println("<br><h2>"+context.getInitParameter("database")+"</h2>");
```

The Servlet API objects covered so far are:

Explicitly defined but instantiation done by Servlet Container

1. HttpServlet object : Custom Servlets
   Ex. public class MyServlet extends HttpServlet { }

Implicitly created

2. HttpServletRequest object : contains information coming from client application(browser, postman, Swagger etc)
3. HttpServletResponse object: contains information that is sent to client from the Server.
4. ServletConfig object
5. ServletContext object

Apart from initialization, we can also **explicitly store data** in the following objects:

1. HttpServletRequest object
2. HttpServletSession object
3. ServletContext object

The following methods:

getAttribute()
setAttribute()
removeAttribute()

```java
package com.wipro.controller;
import java.io.IOException;
import java.io.PrintWriter;
import jakarta.servlet.ServletContext;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
@WebServlet("/attr")
public class AttributeServlet extends HttpServlet {
	private static final long serialVersionUID = 1L;


	protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

		PrintWriter out = response.getWriter();
		request.setAttribute("greeting", "Hi, Welcome to my home");
		request.setAttribute("dinner", "Great meal tonight");

		ServletContext context= this.getServletContext();
		context.setAttribute("college","MGM College of Engineering");


		out.println("<br>"+ request.getAttribute("greeting"));
		out.println("<br>"+ request.getAttribute("dinner"));


		out.println("<br><br>"+ context.getAttribute("college"));

	}

	protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

		doGet(request, response);
	}
}
```

```
package com.wipro.controller;
import java.io.IOException;
import java.io.PrintWriter;
import jakarta.servlet.ServletConfig;
import jakarta.servlet.ServletContext;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebInitParam;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
/*
* http://localhost:8081/dynamic-web-app-demo/hello
*
* Since default HTTP method is GET, control enters into gdoGet() method
*
*    HttpServlet -> GenericServlet -----> Servlet
*
*    Servlet is an interface, GenericServlet is a class that implements Servlet,
*    HttpServlet is class that extends GenericServlet.
*
*
*    GenericServlet can handle any type of protocol( Http, ftp, SMTP etc) whereas
HttpServlet
*    can handle only Http protocol. Since most of the web applications are http-base, we
create our own
*    servlet that extend HttpServlet.
*
*/
@WebServlet(urlPatterns= "/hello" ,
                        initParams = {
                                                @WebInitParam(name = "user", value =
"Srini"),
                                                @WebInitParam(name = "job", value =
"Trainer")
                                        }
```

```
                        )
public class HelloWorldServlet extends HttpServlet {
        private static final long serialVersionUID = 1L;

        protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

                PrintWriter out = response.getWriter();
                //dynamically generate the web page, writes into response object
                out.println("<html><body><h1><font color='red'> Welcome to My Dynamic
Page</font></h1></body></html>");

                //getting reference to ServletConfig object of HelloWorldServlet instance
                ServletConfig config = this.getServletConfig();
                //ServletConfig object contains initialization values of the servlet
                out.println(config.getInitParameter("user"));
                out.println(config.getInitParameter("job"));

                ServletContext context = this.getServletContext();
                out.println("<br><h2>"+context.getInitParameter("database")+"</h2>");

                out.println("<br><h2>"+context.getAttribute("college")+"</h2>");



        }

        protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

                doGet(request, response);
        }
}
```

—----------

JEE- Day2

```
package com.wipro.model;
```

```java
/*
* POJO class: Plain Old Java Object
*/
public class User {
	private String email;
	private String password;

	public User() {

	}

	public User(String email, String password) {
		super();
		this.email = email;
		this.password = password;
	}
	public String getPassword() {
		return password;
	}
	public void setPassword(String password) {
		this.password = password;
	}
	public String getEmail() {
		return email;
	}
	public void setEmail(String email) {
		this.email = email;
	}



}
```

—----------

```java
package com.wipro.controller;
import java.io.IOException;
import java.io.PrintWriter;
```

```java
import com.wipro.model.User;
import jakarta.servlet.ServletContext;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
/*
* If Http method is GET, the form data is appended to the url and sent to the server in the foll. format:
*
*                                   url?querystring
*
*                  query string format:
*                                   name=value&name=value
*
*          Ex.
*
http://localhost:8081/dynamic-web-app-demo/login?userid=Srini&password=Srini%40123
*
*                  While sending sensitive data or large amount of data to the server, apply HttP POST/PUT methods
*/
@WebServlet("/login")
public class LoginServlet extends HttpServlet {
        private static final long serialVersionUID = 1L;


        protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
                try {
                        PrintWriter out = response.getWriter();
//                      String email = request.getParameter("email");
//                      String password =  request.getParameter("password");
//

//                      User user = new User(email,password);
```

```
                        User user = new User(request.getParameter("email"),
request.getParameter("password"));

                        out.println("<html><body><h2>Your Credentials:"+
user.getEmail()+","+user.getPassword()+"</h2></body></html>");
                        ServletContext context = this.getServletContext();

out.println("<br><h2>"+context.getInitParameter("database")+"</h2>");

                }catch(Exception e) {

response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR,e.getMe
ssage());
                }

        }

        protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

                doGet(request, response);
        }
}
```

```
package com.wipro.model;
import java.time.LocalDate;
/*
* POJO class: Plain Old Java Object
*/
public class User {
        private String email;
        private String password;
        private LocalDate birthdate;

        public User() {

        }
```

```
	public User(String email, String password, LocalDate birthdate) {
		super();
		this.email = email;
		this.password = password;
		this.birthdate = birthdate;
	}
	public String getPassword() {
		return password;
	}
	public void setPassword(String password) {
		this.password = password;
	}
	public String getEmail() {
		return email;
	}
	public void setEmail(String email) {
		this.email = email;
	}
	public LocalDate getBirthdate() {
		return birthdate;
	}
	public void setBirthdate(LocalDate birthdate) {
		this.birthdate = birthdate;
	}
	@Override
	public String toString() {
		return "User [email=" + email + ", password=" + password + ", birthdate="
+ birthdate + "]";
	}



}
—-----------------------------

package com.wipro.controller;
import java.io.IOException;
import java.io.PrintWriter;
import java.time.LocalDate;
import com.wipro.model.User;
```

```java
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
/*
* If Http method is GET, the form data is appended to the url and sent to the server in the foll. format:
*
*                               url?querystring
*
*               query string format:
*                               name=value&name=value
*
*       Ex.
*
http://localhost:8081/dynamic-web-app-demo/login?userid=Srini&password=Srini%40123
*
*               While sending sensitive data or large amount of data to the server, apply HttP POST/PUT methods
*
*
*       request.getParameter() return type is String, so when receiving non-string data from the browser
*convert String data to corresponding types.
*/
@WebServlet("/login")
public class LoginServlet extends HttpServlet {
        private static final long serialVersionUID = 1L;


        protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
                try {
                        PrintWriter out = response.getWriter();
//                      String email = request.getParameter("email");
//                      String password =  request.getParameter("password");
//
```

```
//                      User user = new User(email,password);

//                      System.out.println(request.getParameter("birthdate"));
                        //convert String to LocalDate
                        LocalDate birthdate =
LocalDate.parse(request.getParameter("birthdate"));

                        User user = new User(request.getParameter("email"),
                                        request.getParameter("password"), birthdate);

                        out.println("<html><body><h2>Your Credentials:"+
user.getEmail()+","+
                                                                user.getPassword()+","+
user.getBirthdate()+"</h2></body></html>");
//                      ServletContext context = this.getServletContext();
//
out.println("<br><h2>"+context.getInitParameter("database")+"</h2>");

                }catch(Exception e) {

response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR,e.getMe
ssage());
                }

        }

        protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

                doGet(request, response);
        }
}
```

—----------

Inter-Servlet communication

1. Redirection
   response.sendRedirect()

request -> Servlet1 -> redirect- >Servlet2

Servlet1

—----------

response.sendRedirect(“Servlet2”);

Servlet1 will send response object back to the client by updating 2 response headers:

1. status: 302
2. location: “Servlet2”

Now the browser will look into response header and find status code as 302, which it understands this to redirect to url specified in location header

Browser send a new request to the url specified in the location response header

Now request comes to Servlet2 and this servlet sends response back to the browser

In this method, there are 2 request/response cycles involved.

Note: Only the response generated by Servlet2 is rendered to browser

2. Using RequestDispatcher object

There are two ways of accessing RequestDispatcher object.

1. Through ServletRequest object
2. Through ServletContext object

Servlet1

—----------

request.getRequestDispatcher(“url”).forward()

OR

request.getRequestDispatcher(“url”).include()

this.getServletContext().getRequestDispatcher(“/url”).include();

OR

this.getServletContext().getRequestDispatcher("/url").forward();

Note:
If we get reference of RequestDispatcher object through request object, provide relative path of the url.

If we get reference of RequestDispatcher object through ServletContext object, provide absolute path of the url by place / before url.

In this method, there is only 1 request/response cycle

Control will directly go to Servlet2 from Servlet1

request -> Servlet1 -> requestDispatcher.include() -> Servlet2
request -> Servlet1 -> requestDispatcher.forward() -> Servlet2

include() : the response generated by both Servlet1 and Servlet2 is sent to the browser.

forward() : the response generated by only Servlet2 is sent to the browser.

```
package com.wipro.controller;
import java.io.IOException;
import java.io.PrintWriter;
import jakarta.servlet.RequestDispatcher;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
@WebServlet("/servlet1")
public class Servlet1 extends HttpServlet {
	private static final long serialVersionUID = 1L;
```

```
	protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

		PrintWriter out = response.getWriter();
		out.println("Entering  Servlet1...");
		System.out.println(response.getStatus());
		System.out.println(response.getHeader("location"));

		//back to client/browser with response headers, status=302, and
location="Servlet2"
//		response.sendRedirect("Servlet2");

//		RequestDispatcher  requestDispatcher =
request.getRequestDispatcher("Servlet2");
//		requestDispatcher.include(request, response);

		request.getRequestDispatcher("Servlet2").include(request, response);

		System.out.println(response.getStatus());
		System.out.println(response.getHeader("location"));
		out.println("Leaving ServeIt1..");
	}

	protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

		doGet(request, response);
	}
}
```

—----------

```
package com.wipro.controller;
import java.io.IOException;
import java.io.PrintWriter;
import jakarta.servlet.RequestDispatcher;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
```

```
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
@WebServlet("/servlet1")
public class Servlet1 extends HttpServlet {
	private static final long serialVersionUID = 1L;


	protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

		PrintWriter out = response.getWriter();
		out.println("Entering  Servlet1...");
		System.out.println(response.getStatus());
		System.out.println(response.getHeader("location"));

		//back to client/browser with response headers, status=302, and
location="Servlet2"
//		response.sendRedirect("Servlet2");

//		RequestDispatcher  requestDispatcher =
request.getRequestDispatcher("Servlet2");
//		requestDispatcher.include(request, response);

//		request.getRequestDispatcher("servlet2").include(request, response);
//		request.getRequestDispatcher("servlet2").forward(request, response);



this.getServletContext().getRequestDispatcher("/servlet2").include(request, response);
//
this.getServletContext().getRequestDispatcher("/servlet2").forward(request, response);

		System.out.println(response.getStatus());
		System.out.println(response.getHeader("location"));
		out.println("Leaving ServeIt1..");
	}

	protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
```

```
		doGet(request, response);
	}
}
```

—----------

```
package com.wipro.controller;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

@WebServlet("/servlet2")
public class Servlet2 extends HttpServlet {
	private static final long serialVersionUID = 1L;


	protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
		PrintWriter out = response.getWriter();

		out.println("Entering  Servlet2...");
		System.out.println(response.getStatus());
		out.println("Leaving ServeIt2..");
	}


	protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
		// TODO Auto-generated method stub
		doGet(request, response);
	}

}
```

—----------

Session Handling

Which protocol is used in communication between client/browser and sever ?
HTTP

That is why the client is also called as HTTP client and Server is called HTTP Server.

HTTP is a stateless protocol.

The request coming from the HTTP client i.e browser is always interpreted as a new request to the Server. I.e Server doesn't remember the previous interaction made with the client.

Session tracking is a way to maintain the HTTP client state between multiple request/response cycles that take place between HTTP client/browser and the Server.

In most real-time web applications, maintaining state between multiple request/response cycles between the client and the server is required.

Ex. Online Reservation Apps, Online Shopping Apps, Online banking Apps etc.

To maintain state , following method are used:

1. Cookies
   A cookie is a small text file created by the server and sent to the client/browser. The cookie file contains session data in form of name:value pairs.

   Clients can disable cookies.

   Then how to maintain the state ? Another approach is adapted and that is URL-Rewriting.

Server once knowing that the client doesn't support cookies, it will append the cookie to the url as query string and sent to the browser.

The browser in the subsequent request will also append the cookie to the url as query string and send to the browser.

Url?cookie

Server-side session management:

Server will maintain the client state i.e. session data in an object and and send reference of that object to the browser.

The cookie file will contain only the identifier,

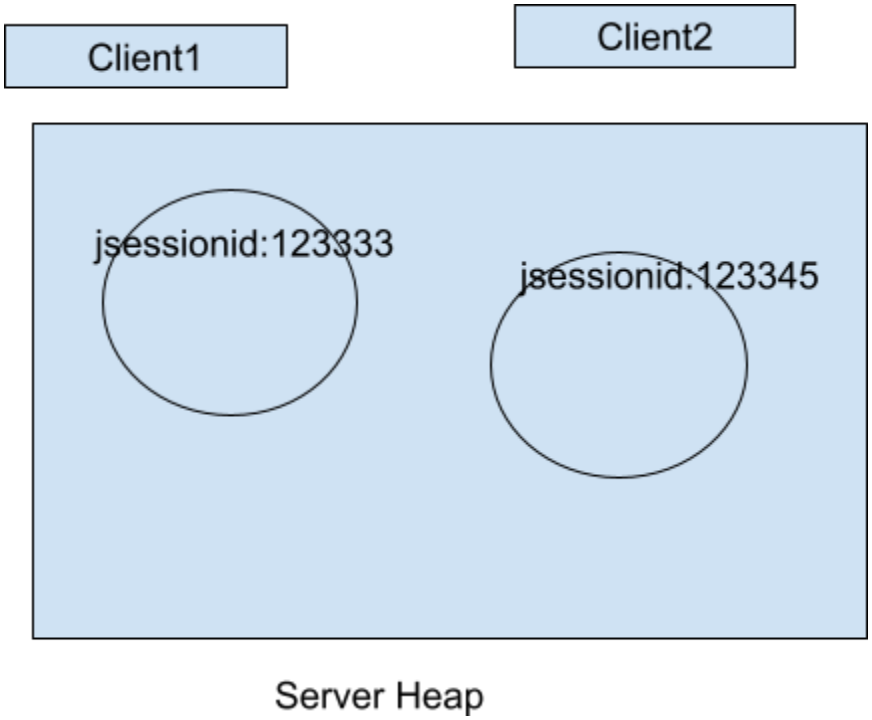
Ex. jsessionid =46816283923

If cookie is disabled, the URL is rewritten by append the identifier as foll..

url?jsessionid=572672999

Server

—----------



Servlet API has an interface, HttpSession

request.getSession() returns the reference of the HttpSession object.

```
package com.wipro.controller;
import java.io.IOException;
import java.io.PrintWriter;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;
@WebServlet("/controller")
public class Controller extends HttpServlet {
	private static final long serialVersionUID = 1L;

	protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
```

```
		PrintWriter out = response.getWriter();

		/*
		 * creates HttpSession object and return its reference, it will create cookie file, place jsessionid=XXXX
		 * in the cookie file and send it to client.
		 *
		 * request.getSession(): create a new session object if  not existing, if existing returns its reference
		 */
		HttpSession session = request.getSession();
		System.out.println(session.getId());

		session.setAttribute("email", "digitech1993@gmail.com");

		request.getRequestDispatcher("servlet1").forward(request, response);



	}

	protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

		doGet(request, response);
	}
}
```

—----------

```
package com.wipro.controller;
import java.io.IOException;
import java.io.PrintWriter;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;
```

```java
@WebServlet("/servlet1")
public class Servlet1 extends HttpServlet {

	protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
		/*
		 * getSession(false) , returns id if session object is already existing,else returns null
		 */
		PrintWriter out = response.getWriter();

		HttpSession  session = request.getSession(false);
		out.println(session.getId());

		out.println(session.getAttribute("email"));
	}

	protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

		doGet(request, response);
	}
}
```

—----------

Filters

One or more filters can be placed between the client(request) and controller(servlet).

Before the request object hits the controller, we can place one or more filters in between.

Why?

Filters can be used for authentication, request validations

The response also has to go through the filter before it is rendered to the client.

Filters can perform encryption, modification, auditing

Filters will perform pre-processing
client->request ->filter1->filter2->filter3->servlet

Filters will perform post-processing
servlet ->response -> filter3 -> filter2 -> filter1 -> client

—----------

```
package com.wipro.filter;
import jakarta.servlet.Filter;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.ServletRequest;
import jakarta.servlet.ServletResponse;
import jakarta.servlet.annotation.WebFilter;
import jakarta.servlet.http.HttpFilter;
import java.io.IOException;
@WebFilter(filterName= "/filter1", urlPatterns="/controller")
public class ValidateFilter extends HttpFilter implements Filter {


	public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain) throws IOException, ServletException {

		System.out.println("request object entered into ValidateFilter");
		// pass the request along the filter chain
		chain.doFilter(request, response);
		System.out.println("response object exiting ValidateFilter");
	}

}
```

—----------

```
package com.wipro.filter;
import jakarta.servlet.Filter;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebFilter;
import jakarta.servlet.http.HttpFilter;
```

```
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;
@WebFilter(filterName="/filter2", urlPatterns="/controller")
public class AuthenticateFilter extends HttpFilter implements Filter {

	public void doFilter(HttpServletRequest request, HttpServletResponse
response, FilterChain chain) throws IOException, ServletException {
		System.out.println("Entered Filter2");
		chain.doFilter(request, response);
		System.out.println("Exiting Filter2");
	}

}
```

—----------

```
package com.wipro.controller;
import java.io.IOException;
import java.io.PrintWriter;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;
@WebServlet("/controller")
public class Controller extends HttpServlet {
	private static final long serialVersionUID = 1L;

	protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

		PrintWriter out = response.getWriter();

		/*
		 * creates HttpSession object and return its reference, it will create cookie
file, place jsessionid=XXXX
		 * in the cookie file and send it to client.
		 *
```

```
		 * request.getSession(): create a new session object if  not existing, if existing returns its reference
		 */
		HttpSession session = request.getSession();
		System.out.println(session.getId());

		session.setAttribute("email", "digitech1993@gmail.com");

		request.getRequestDispatcher("servlet1").forward(request, response);



	}

	protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

		doGet(request, response);
	}
}
```

—————————————

```
package com.wipro.controller;
import java.io.IOException;
import java.io.PrintWriter;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;
@WebServlet(urlPatterns="/servlet1")
public class Servlet1 extends HttpServlet {

	protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
		/*
```

```
                * getSession(false) , returns id if session object is already existing,else
returns null
                */
                PrintWriter out = response.getWriter();

                HttpSession  session = request.getSession(false);
                out.println(session.getId());

                out.println(session.getAttribute("email"));
        }

        protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

                doGet(request, response);
        }
}
```

—----------

JSP

Tags in JSP can be categorized into:
•Comments

•Scripting Elements
- Declarations
- Expressions
- Scriptlets

•Directive Elements

page directive

Ex.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
  pageEncoding="UTF-8"  import="java.time.*" %>
```

include directive

taglib directive

•Action Elements

—----------demo.jsp—---------

```
<h3>Hello, Today's date is</h3>

        <!--
                Java code is placed within a scriptlet.

                what is out? out is an implicit object i.e JSP provides PrintWriter object
called out by default.
                There are 9 implicit objects which we can use in JSP.

                                out, request, response, config, session, application,
exception, page, pageContext


                                In the above, config is nothing but ServletConfig object
                                application is ServletContext object
        -->
        <%

                LocalDateTime  today = LocalDateTime.now();
                out.println(today);
                request.setAttribute("email","digitech1993@gmail.com");


        %>

        <h3>My Email Id  is: </h3>
        <%
                out.println(request.getAttribute("email")) ;
        %>
        <h4> End of the Page</h4>
```

—----------

sample.jsp -> sample_jsp.java

```
<h1> Sample Java Server Page</h1>
        <%-- Declaration , inside servlet , outside methods--%>
        <%!
                int counter=0;

                public void greeting(){
                        System.out.println("Good Day, Now the time is: " +
LocalTime.now());
                }
        %>

        <%-- Expression --%>
        <%= LocalDateTime.now()  %>


        <%-- Scriptlet --%>
        <%

                out.println("I'm A JSP Page");
        %>

        <h4> **********************************</h4>

        ====================================

Following translation to .java is done automatically

public class Sample_Jsp extends HttpServlet{
                int counter=0; // instance variable
                //instance method
                public void greeting(){
                        System.out.println("Good Day, Now the time is: " +
LocalTime.now());
                }



                public void _jspService(HttpServletRequest request, HttpServletResponse
                                response){
                        PrintWriter out = response.getWriter();
```

```
                out.println("<h1> Sample Java Server Page</h1>");
                out.println(LocalDateTime.now() );
                out.println("I'm A JSP Page");
                out.println("<h4> ******************************</h4>");

        }



}
```

——————sample.jsp——————

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"  import="java.time.*"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
        <h1> Sample Java Server Page</h1>
        <%-- Declaration --%>
        <%!
                int counter=0;

                public void greeting(){
                        System.out.println("Good Day, Now the time is: " +
LocalTime.now());
                }
        %>

        <%-- Expression --%>
        <%= LocalDateTime.now()  %>
```

```
<%-- Scriptlet --%>
<%

        out.println("I'm A JSP Page");
%>

<h4> **********</h4>


</body>
</html>
```

JSP directive:
There are 3 JSP directives, page, include and taglib
JSP directive begin with **<%@** and end with **%>**

Ex. page directive

```
<%@ page language="java" con  tentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"  import="java.time.*"%>
```

include directive

To include one jsp file into another jsp file.

Ex. Say we want some header and footer information for all the JSP pages.

header.jsp
footer.jsp

page1.jsp
page2.jsp
…

page1.jsp

—----------

<%@ include file=”header.jsp” %>

Page1 contents

<%@ include file=”footer.jsp” %>

During translation phase,

<%@ include file=”header.jsp” %>
Is replaced with its contents

And
<%@ include file=”footer.jsp” %>
Is replaced with its contents.

Same thing we repeat for other pages

—----------header.jsp—---------

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
  pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
        <h1>Introduction to Spring Framework</h1>
        <hr>
</body>
</html>
```

—----------footer.jsp—----------

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
  pageEncoding="UTF-8"%>
```

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
	<hr>
	<h4 align='center'>copyright 2024@Wipro Ltd.</h4>
</body>
</html>

—----------------------demo.jsp—----------

<%@ page language="java" contentType="text/html; charset=UTF-8"
  pageEncoding="UTF-8"  import="java.time.*" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Sample Java Server Page</title>
</head>
<body>
	<%@ include file="header.jsp" %>
	<h3>Hello, Today's date is</h3>

	<!--
		Java code is placed within a scriptlet.

		what is out? out is an implicit object i.e JSP provides PrintWriter
object called out by default.
		There are 9 implicit objects which we can use in JSP.

				out, request, response,  config,  session,   application,
exception, page, pageContext


				In the above, config is nothing but ServletConfig object
				application is ServletContext object
	-->
```

```
	<%

		LocalDateTime  today = LocalDateTime.now();
		out.println(today);
		request.setAttribute("email","digitech1993@gmail.com");


	%>

	<h3>My Email Id  is: </h3>
	<%
		out.println(request.getAttribute("email")) ;
	%>

	<%@ include file="footer.jsp" %>
</body>
</html>


—----------------------sample.jsp—----------------
<%@ page language="java" contentType="text/html; charset=UTF-8"
  pageEncoding="UTF-8"  import="java.time.*"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
	<%@ include file="header.jsp" %>
	<h1> Sample Java Server Page</h1>
	<%-- Declaration --%>
	<%!
		int counter=0;

		public void greeting(){
			System.out.println("Good Day, Now the time is: " +
LocalTime.now());
		}
	%>
```

```
<%-- Expression --%>
<%= LocalDateTime.now()  %>s


<%-- Scriptlet --%>
<%

        out.println("I'm A JSP Page");
%>

<%@ include file="footer.jsp" %>



</body>
</html>
```

—----------

3. taglib directive

To include custom JSP tags from other libraries

We have third-party library called, jstl.
We can include this library into our JSP file and use the tags in it.

```
<dependency>
        <groupId>jakarta.servlet.jsp.jstl</groupId>
        <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
        <version>3.0.0</version>
</dependency>
<dependency>
        <groupId>org.glassfish.web</groupId>
        <artifactId>jakarta.servlet.jsp.jstl</artifactId>
        <version>3.0.1</version>
</dependency>
```

—----------index.jsp—----------

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
  pageEncoding="UTF-8"  import="java.util.*" isELIgnored = "false"  %>

<%@ taglib prefix="c" uri="jakarta.tags.core" %>
<html>
<body>
        <h2>JSTL Demo</h2>
        <c:set var="income" scope="session" value="${4000*4}"/>

        <p>Your income is : <c:out value="${income}"/> </p>
_____
_____
        <c:choose>
                <c:when test="${income <= 10000}">
                        Income is not good.
                </c:when>
                <c:when test="${income > 20000}">
                        Income is very good.
                </c:when>
                <c:otherwise>
                        Income is undetermined...
                </c:otherwise>
        </c:choose>
        <%
                Map<String, String> countryCapitalMap = new HashMap<>();
                countryCapitalMap.put("United States", "Washington DC");
                countryCapitalMap.put("India",
"Delhi");countryCapitalMap.put("Germany", "Berlin");
                countryCapitalMap.put("France",
"Paris");countryCapitalMap.put("Italy", "Rome");
                request.setAttribute("capital", countryCapitalMap);
        %>
        <%--JSP Code --%>
        <table border="1">
                <tr><th bgcolor="green">COUNTRY</th><th
bgcolor="green">CAPITAL</th></tr>
                <c:forEach var="c" items="${capital}">
                        <tr><td>${c.key}</td><td>${c.value}</td></tr>
```

```
            </c:forEach>
        </table>
</body>
</html>


—----------------------------------------------------------------------

<%@ page language="java" contentType="text/html; charset=UTF-8"
  pageEncoding="UTF-8"  import="java.util.*" isELIgnored = "false"  %>

<%@ taglib prefix="c" uri="jakarta.tags.core" %>
<html>
<body>
        <h2>JSTL Demo</h2>
        <c:set var="income" scope="session" value="${12000}"/>

        <p>Your income is : <c:out value="${income}"/> </p>
_____
_____
_____
_____<c:choose>
                <c:when test="${income <= 10000}">
                        Income is not good.
                </c:when>
                <c:when test="${income > 20000}">
                        Income is very good.
                </c:when>
                <c:otherwise>
                        Income is undetermined...
                </c:otherwise>
        </c:choose>
        <%
_____________Map<String, String> countryCapitalMap = new HashMap<>();
_____________countryCapitalMap.put("United States", "Washington DC");
_____________countryCapitalMap.put("India",
"Delhi");countryCapitalMap.put("Germany", "Berlin");
_____________countryCapitalMap.put("France",
"Paris");countryCapitalMap.put("Italy", "Rome");
_____________
```

```
            request.setAttribute("capital", countryCapitalMap);

            List<String> courseList = new ArrayList<>();
            courseList.add("Java"); courseList.add("Java EE");
            courseList.add("Javascript");courseList.add("Spring");
            courseList.add("JPA");courseList.add("Microservices");

            request.setAttribute("courseList",courseList);
        %>
        <%--JSP Code --%>
        <table border="1">
            <tr><th bgcolor="green">COUNTRY</th><th
bgcolor="green">CAPITAL</th></tr>
            <c:forEach var="c" items="${capital}">
                <tr><td>${c.key}</td><td>${c.value}</td></tr>
            </c:forEach>
        </table>

        </br>

        <table border="1">
            <tr><th bgcolor="green">Course Name</th></tr>
            <c:forEach var="i" items="${courseList}">
                <tr><td>${i}</td></tr>
            </c:forEach>
        </table>

</body>
</html>
```

—----------

JSP Action Tags

<jsp:include> same as RequestDispatcher object's include() method
<jsp:forward> same as RequestDispatcher object's forward() method

<jsp:useBean>

Purpose: To instantiate Java Bean.

A bean is Java POJO class.

<jsp:setProperty> and <jsp:getProperty>

—----------Person.java—---------

```
package com.wipro.model;
public class Person {
	private Long adharCard;
	private String firstName;
	private String lastName;
	private String address;
	private Long mobile;


	public Person() {

	}
	public Person(Long adharCard, String firstName, String lastName, String
address, Long mobile) {
		super();
		this.adharCard = adharCard;
		this.firstName = firstName;
		this.lastName = lastName;
		this.address = address;
		this.mobile = mobile;
	}
	public Long getAdharCard() {
		return adharCard;
	}
	public void setAdharCard(Long adharCard) {
		this.adharCard = adharCard;
	}
	public String getFirstName() {
		return firstName;
	}
	public void setFirstName(String firstName) {
```

```
		this.firstName = firstName;
	}
	public String getLastName() {
		return lastName;
	}
	public void setLastName(String lastName) {
		this.lastName = lastName;
	}
	public String getAddress() {
		return address;
	}
	public void setAddress(String address) {
		this.address = address;
	}
	public Long getMobile() {
		return mobile;
	}
	public void setMobile(Long mobile) {
		this.mobile = mobile;
	}
	@Override
	public String toString() {
		return "Person [adharCard=" + adharCard + ", firstName=" +
firstName + ", lastName=" + lastName + ", address="
				+ address + ", mobile=" + mobile + "]";
	}



}
```

—----------person-details.jsp----------

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
  pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
```

```
<meta charset="UTF-8">
<title>Person Details Page</title>
</head>
<body>

	<%-- To instantiate Person class, use <jsp:useBean> tag
			same as  Person personBean = new Person();
	 --%>

	<jsp:useBean id="personBean"  class="com.wipro.model.Person"
scope="request"/>

	<%-- To set properties, use <jsp:setProperty> tag --%>
	<jsp:setProperty property="adharCard" name="personBean"
value="678765786923"/>
	<jsp:setProperty property="firstName" name="personBean" value="Ravi"/>
	<jsp:setProperty property="lastName" name="personBean"
value="Sharma"/>
	<jsp:setProperty property="address" name="personBean" value="Pune"/>
	<jsp:setProperty property="mobile" name="personBean"
value="8976567658"/>


	<h2> Person Details</h2>
	<b>Adhar Card:</b> ${personBean.adharCard }<br>
	<b>First Name:</b> ${personBean.firstName }<br>
	<b>Last Name:</b> ${personBean.lastName }<br>
	<b>Address:</b> ${personBean.address }<br>
	<b>Mobile:</b> ${personBean.mobile }<br>



</body>
</html>
```

—----------

Spring Framwork

```java
package com.wipro.model;
public class Engine {
	private Long serialNumber;
	private Double capacity;
	private String type;


	public Engine() {

	}
	public Engine(Long serialNumber, Double capacity, String type) {
		super();
		this.serialNumber = serialNumber;
		this.capacity = capacity;
		this.type = type;
	}
	public Long getSerialNumber() {
		return serialNumber;
	}
	public void setSerialNumber(Long serialNumber) {
		this.serialNumber = serialNumber;
	}
	public Double getCapacity() {
		return capacity;
	}
	public void setCapacity(Double capacity) {
		this.capacity = capacity;
	}
	public String getType() {
		return type;
	}
	public void setType(String type) {
		this.type = type;
	}
	@Override
	public String toString() {
		return "Engine [serialNumber=" + serialNumber + ", capacity=" +
capacity + ", type=" + type + "]";
```

```
	}


}

—----------------

package com.wipro.model;
/*
* Car has-a Engine.
*
* Her Car is dependent and Engine is dependency
*
* How to inject dependency (Engine object) into dependent(Car object) ?
*/
public class Car {
	private String brand;
	private String model;
	private Engine engine;



	public Car() {

	}
	//Constructor injection
	public Car(String brand, String model, Engine engine) {
		super();
		this.brand = brand;
		this.model = model;
		this.engine = engine;
	}
	public String getBrand() {
		return brand;
	}
	public void setBrand(String brand) {
		this.brand = brand;
	}
	public String getModel() {
		return model;
```

```
	}
	public void setModel(String model) {
		this.model = model;
	}
	public Engine getEngine() {
		return engine;
	}
	//setter injection
	public void setEngine(Engine engine) {
		this.engine = engine;
	}




}

—----------------------------------------
package com.wipro.app;
import com.wipro.model.Car;
import com.wipro.model.Engine;
/*
* Here, the application, CarPurchase is explicitly instantiating, initializing , injecting
* and destroying the dependencies.
*
* There strong bonding between the application and the dependencies which is not
* Suggestible, instead recommended concept is loose-coupling.
*
* How to achieve loose coupling?
* Instead of application or dependent object managing the life-cycle of dependencies,
* hand over this responsibility to some third-party so that the third-party will manage
* the dependencies.
*
* This third-party can be Spring or Struts or JSF and so on.
*
```

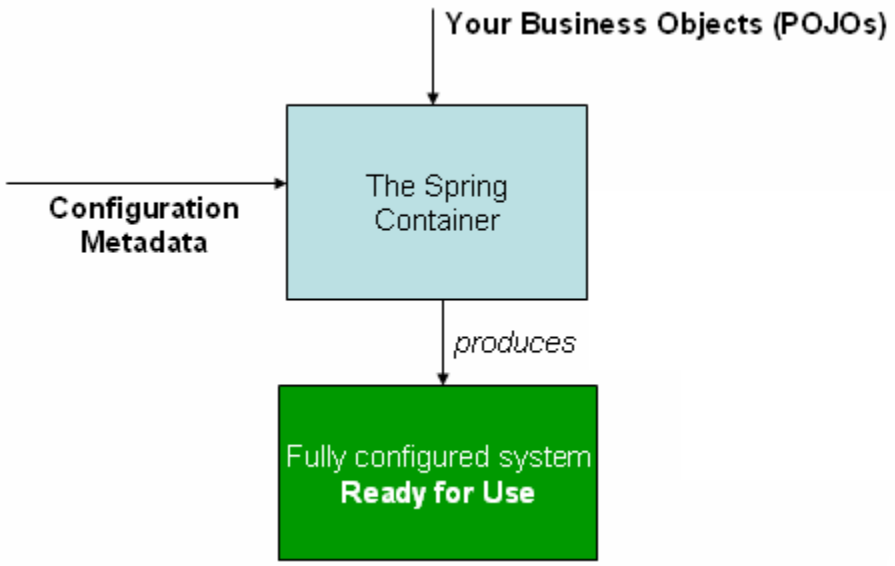
```
* Spring container manages the dependencies. Earlier it was dependent taking care of all these, now
* the control of managing the dependencies to handed over to a third-party ie. control has been
* inverted hence the term inversion-control.
*
* Dependency injection (DI) is a specialized form of IoC, whereby objects define their dependencies
* (that is, the other objects they work with) only through constructor arguments, arguments to a factory method,
*  or properties that are set on the object instance after it is constructed or returned from a factory method.
* The IoC container then injects those dependencies when it creates the bean.
*
*/

public class CarPurchase {
	public static void main(String[] args) {
		//dependency
		Engine  engine = new Engine(12345678L,2700.0,"petrol");

		//constructor injection
		Car  myCar = new Car("Maruthi","Grand Vitara",engine);

		System.out.println(myCar);


		Car myAnotherCar = new Car();

		//setter injection
		myAnotherCar.setEngine(engine);
		myAnotherCar.setBrand("Hyundai");
		myAnotherCar.setModel("Verna");


		engine=null;

		System.out.println(myAnotherCar);
	}
}
```

—----------

The `org.springframework.beans` and `org.springframework.context` packages are the basis for Spring Framework's IoC container. The `BeanFactory` interface provides an advanced configuration mechanism capable of managing any type of object. `ApplicationContext` is a sub-interface of `BeanFactory`. It adds:

- **Easier integration with Spring's AOP features**
- **Message resource handling (for use in internationalization)**
- **Event publication**
- **Application-layer specific contexts such as the `WebApplicationContext` for use in web applications.**

In short, the `BeanFactory` provides the configuration framework and basic functionality, and the `ApplicationContext` adds more enterprise-specific functionality.



XML as an External Configuration DSL

XML-based configuration metadata configures these beans as `<bean/>` elements inside a top-level `<beans/>` element. The following example shows the basic structure of XML-based configuration metadata:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="
        http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">
<!-- bean definitions here -->

</beans>
```

—----------

<!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->
<dependency>
** <groupId>org.springframework</groupId>**
** <artifactId>spring-context</artifactId>**
** <version>6.1.8</version>**
</dependency>

—----------

```
public class App {

        public static void main(String[] args) {
```

```
		ApplicationContext context=null;
		try {
			//IoC container
			context= new ClassPathXmlApplicationContext("spring.xml");




			 //shutdown IoC

((AbstractApplicationContext)context).registerShutdownHook();
		}catch(Exception e) {
			e.printStackTrace();
		}finally {
			((AbstractApplicationContext)context).close();
		}

	}

}
```

—----------

https://docs.spring.io/spring-framework/docs/4.2.x/spring-framework-reference/html/xsd-configuration.html

xml class
empno -> setEmpno()
ename -> setEname()
customHiredate -> setCustomHiredate()

—----------

```
package com.wipro.model;
public class Address {
	private String doorNumber;
	private String street;
	private String locality;
	private String city;
	private Long pincode;
```

```
	public Address() {

	}
	public Address(String doorNumber, String street, String locality, String
city, Long pincode) {
		super();
		this.doorNumber = doorNumber;
		this.street = street;
		this.locality = locality;
		this.city = city;
		this.pincode = pincode;
	}
	public String getDoorNumber() {
		return doorNumber;
	}
	public void setDoorNumber(String doorNumber) {
		this.doorNumber = doorNumber;
	}
	public String getStreet() {
		return street;
	}
	public void setStreet(String street) {
		this.street = street;
	}
	public String getLocality() {
		return locality;
	}
	public void setLocality(String locality) {
		this.locality = locality;
	}
	public String getCity() {
		return city;
	}
	public void setCity(String city) {
		this.city = city;
	}
	public Long getPincode() {
		return pincode;
```

```
	}
	public void setPincode(Long pincode) {
		this.pincode = pincode;
	}
	@Override
	public String toString() {
		return "Address [doorNumber=" + doorNumber + ", street=" + street
+ ", locality=" + locality + ", city=" + city
				+ ", pincode=" + pincode + "]";
	}




}

—----------------------------------------------------------------
package com.wipro.model;
import java.time.LocalDate;
public class Person {
	private Long adharCard;
	private String name;
	private LocalDate birthdate;
	private Address temporaryAddress;
	private Address permanentAddress;


	public Person() {

	}
	public Person(Long adharCard, String name, LocalDate birthdate, Address
temporaryAddress,
			Address permanentAddress) {
		super();
		this.adharCard = adharCard;
		this.name = name;
		this.birthdate = birthdate;
		this.temporaryAddress = temporaryAddress;
		this.permanentAddress = permanentAddress;
	}
```

```
	public Person(Long adharCard, String name, String birthdate, Address
temporaryAddress,
			Address permanentAddress) {
		super();
		this.adharCard = adharCard;
		this.name = name;
		this.birthdate = LocalDate.parse(birthdate);
		this.temporaryAddress = temporaryAddress;
		this.permanentAddress = permanentAddress;
	}
	public Long getAdharCard() {
		return adharCard;
	}
	public void setAdharCard(Long adharCard) {
		this.adharCard = adharCard;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public LocalDate getBirthdate() {
		return birthdate;
	}
	public void setBirthdate(LocalDate birthdate) {
		this.birthdate = birthdate;
	}
	public void setCustomBirthdate(String birthdate) {
		this.birthdate = LocalDate.parse(birthdate);
	}
	public Address getTemporaryAddress() {
		return temporaryAddress;
	}
	public void setTemporaryAddress(Address temporaryAddress) {
		this.temporaryAddress = temporaryAddress;
	}
	public Address getPermanentAddress() {
```

```
		return permanentAddress;
	}
	public void setPermanentAddress(Address permanentAddress) {
		this.permanentAddress = permanentAddress;
	}
	@Override
	public String toString() {
		return "Person [adharCard=" + adharCard + ", name=" + name + ",
birthdate=" + birthdate + ", temporaryAddress="
				+ temporaryAddress + ", permanentAddress=" +
permanentAddress + "]";
	}




}

—----------------------------------------------

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="
    http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd"> <!-- bean
definitions here -->
	<bean id="addressBean1" class="com.wipro.model.Address">
		<constructor-arg name="doorNumber" value="3-4-356"/>
		<constructor-arg name="street" value="MG Street"/>
		<constructor-arg name="locality" value="Abids"/>
		<constructor-arg name="city" value="Hyderabad"/>
		<constructor-arg name="pincode" value="500001"/>
	</bean>

	<bean id="addressBean2" class="com.wipro.model.Address">
		<property name="doorNumber" value="8-6-112"/>
```

```
            <property name="street" value="Kings Street"/>
            <property name="locality" value="Bandra"/>
            <property name="city" value="Mumbai"/>
            <property name="pincode" value="400001"/>
        </bean>

        <!--  Inject the above address beans into person bean, use ref attribute -->

        <bean id="personBean" class="com.wipro.model.Person">
            <property name="adharCard" value="988612548769"/>
            <property name="name" value="Vinod Kumar"/>
            <property name="customBirthdate" value="1990-10-15"/>
            <property name="temporaryAddress" ref="addressBean1" />
            <property name="permanentAddress" ref="addressBean2" />
        </bean>

</beans>

—----------------------------------------
package com.wipro.app;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.wipro.model.Person;
public class App {
        public static void main(String[] args) {
            ApplicationContext context=null;
            try {
                //IoC container
                context= new
ClassPathXmlApplicationContext("spring1.xml");

                Person person = (Person) context.getBean("personBean");
                System.out.println(person);


                 //shutdown IoC

((AbstractApplicationContext)context).registerShutdownHook();
            }catch(Exception e) {
```

```
                e.printStackTrace();
            }finally {
                ((AbstractApplicationContext)context).close();
            }
        }
}
```

Autowiring in xml file:

byName: If the bean id name is same as the property name, automatically inject the bean into the property field.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="
    http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd"> <!-- bean
definitions here -->
        <bean id="temporaryAddress" class="com.wipro.model.Address">
            <constructor-arg name="doorNumber" value="3-4-356"/>
            <constructor-arg name="street" value="MG Street"/>
            <constructor-arg name="locality" value="Abids"/>
            <constructor-arg name="city" value="Hyderabad"/>
            <constructor-arg name="pincode" value="500001"/>
        </bean>

        <bean id="permanentAddress" class="com.wipro.model.Address">
            <property name="doorNumber" value="8-6-112"/>
            <property name="street" value="Kings Street"/>
            <property name="locality" value="Bandra"/>
            <property name="city" value="Mumbai"/>
            <property name="pincode" value="400001"/>
        </bean>
```

```
        <!--  Inject the above address beans into person bean, use ref attribute -->

        <bean id="personBean" class="com.wipro.model.Person"
autowire="byName">
                <property name="adharCard" value="988612548769"/>
                <property name="name" value="Vinod Kumar"/>
                <property name="customBirthdate" value="1990-10-15"/>
                <!-- <property name="temporaryAddress" ref="addressBean1" />
                <property name="permanentAddress" ref="addressBean2" /> -->
        </bean>

</beans>
```

—----------

byType: If the bean type is matching with the datatype of the property, then the bean is
automatically injected into the property field.

—--------spring2.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"
   xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="
       http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd"> <!-- bean
definitions here -->


        <bean id="addressBean" class="com.wipro.model.Address">
                <constructor-arg name="doorNumber" value="3-4-356"/>
                <constructor-arg name="street" value="MG Street"/>
                <constructor-arg name="locality" value="Abids"/>
                <constructor-arg name="city" value="Hyderabad"/>
                <constructor-arg name="pincode" value="500001"/>
```

```
	</bean>

	<!-- <bean id="permanentAddress" class="com.wipro.model.Address">
		<property name="doorNumber" value="8-6-112"/>
		<property name="street" value="Kings Street"/>
		<property name="locality" value="Bandra"/>
		<property name="city" value="Mumbai"/>
		<property name="pincode" value="400001"/>
	</bean> -->

	<!--  Inject the above address beans into person bean, use ref attribute -->

	<bean id="personBean" class="com.wipro.model.Person"
autowire="byType">
		<property name="adharCard" value="988612548769"/>
		<property name="name" value="Vinod Kumar"/>
		<property name="customBirthdate" value="1990-10-15"/>
		<!-- <property name="temporaryAddress" ref="addressBean1" />
		<property name="permanentAddress" ref="addressBean2" /> -->
	</bean>

</beans>


—----------------------------------------
package com.wipro.app;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.wipro.model.Person;
public class App {
	public static void main(String[] args) {
		ApplicationContext context=null;
		try {
			//IoC container
//			context= new
ClassPathXmlApplicationContext("spring1.xml");
			context= new
ClassPathXmlApplicationContext("spring2.xml");
```

```
                Person person = (Person) context.getBean("personBean");
                System.out.println(person);


                 //shutdown IoC

((AbstractApplicationContext)context).registerShutdownHook();
            }catch(Exception e) {
                e.printStackTrace();
            }finally {
                ((AbstractApplicationContext)context).close();
            }
        }
}
```

—----------

Autowiring using Annotations

Instead of using autowire attribute of bean in the xml file, we can use annotation, @Autowired applied to property.

The annotation, @Autowired is byType by default.

—----------

```
package com.wipro.model;
import java.time.LocalDate;
import org.springframework.beans.factory.annotation.Autowired;
public class Person {
        private Long adharCard;
        private String name;
        private LocalDate birthdate;

        @Autowired
        private Address temporaryAddress;
        @Autowired
        private Address permanentAddress;


        public Person() {
```

```
	}
	public Person(Long adharCard, String name, LocalDate birthdate, Address
temporaryAddress,
			Address permanentAddress) {
		super();
		this.adharCard = adharCard;
		this.name = name;
		this.birthdate = birthdate;
		this.temporaryAddress = temporaryAddress;
		this.permanentAddress = permanentAddress;
	}


	public Person(Long adharCard, String name, String birthdate, Address
temporaryAddress,
			Address permanentAddress) {
		super();
		this.adharCard = adharCard;
		this.name = name;
		this.birthdate = LocalDate.parse(birthdate);
		this.temporaryAddress = temporaryAddress;
		this.permanentAddress = permanentAddress;
	}
	public Long getAdharCard() {
		return adharCard;
	}
	public void setAdharCard(Long adharCard) {
		this.adharCard = adharCard;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public LocalDate getBirthdate() {
		return birthdate;
	}
	public void setBirthdate(LocalDate birthdate) {
```

```
		this.birthdate = birthdate;
	}
	public void setCustomBirthdate(String birthdate) {
		this.birthdate = LocalDate.parse(birthdate);
	}
	public Address getTemporaryAddress() {
		return temporaryAddress;
	}
	public void setTemporaryAddress(Address temporaryAddress) {
		this.temporaryAddress = temporaryAddress;
	}
	public Address getPermanentAddress() {
		return permanentAddress;
	}
	public void setPermanentAddress(Address permanentAddress) {
		this.permanentAddress = permanentAddress;
	}
	@Override
	public String toString() {
		return "Person [adharCard=" + adharCard + ", name=" + name + ",
birthdate=" + birthdate + ", temporaryAddress="
				+ temporaryAddress + ", permanentAddress=" +
permanentAddress + "]";
	}



}
```

—----------spring3.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"
  xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="
    http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd"> <!-- bean
definitions here -->
```

```
	<context:annotation-config/>

	<bean id="addressBean1" class="com.wipro.model.Address">
		<constructor-arg name="doorNumber" value="3-4-356"/>
		<constructor-arg name="street" value="MG Street"/>
		<constructor-arg name="locality" value="Abids"/>
		<constructor-arg name="city" value="Hyderabad"/>
		<constructor-arg name="pincode" value="500001"/>
	</bean>

	<!-- <bean id="addressBean2" class="com.wipro.model.Address">
		<property name="doorNumber" value="8-6-112"/>
		<property name="street" value="Kings Street"/>
		<property name="locality" value="Bandra"/>
		<property name="city" value="Mumbai"/>
		<property name="pincode" value="400001"/>
	</bean>  -->

	<!--  Inject the above address beans into person bean, use ref attribute -->

	<bean id="personBean" class="com.wipro.model.Person">
		<property name="adharCard" value="988612548769"/>
		<property name="name" value="Vinod Kumar"/>
		<property name="customBirthdate" value="1990-10-15"/>

	</bean>

</beans>

—----------App.java—-----------------
package com.wipro.app;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.wipro.model.Person;
public class App {
	public static void main(String[] args) {
		ApplicationContext context=null;
		try {
			//IoC container
```

```
//                      context= new
ClassPathXmlApplicationContext("spring1.xml");
//                      context= new
ClassPathXmlApplicationContext("spring2.xml");
                        context= new
ClassPathXmlApplicationContext("spring3.xml");

                        Person person = (Person) context.getBean("personBean");
                        System.out.println(person);


                         //shutdown IoC

((AbstractApplicationContext)context).registerShutdownHook();
                }catch(Exception e) {
                        e.printStackTrace();
                }finally {
                        ((AbstractApplicationContext)context).close();
                }
        }
}
```

—----------

How to change @Autowired from default byType to byName ?

```
@Autowired(required = false)
@Qualifier(value = "BeanId name")
```

—----------Person.java----------

```
package com.wipro.model;
import java.time.LocalDate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
public class Person {
        private Long adharCard;
        private String name;
        private LocalDate birthdate;
```

```
	//The @Autowired is byType   by default
//	@Autowired
//	private Address temporaryAddress;
//	@Autowired
//	private Address permanentAddress;


	// The @Autowired changed to byName
	@Autowired(required = false)
	@Qualifier(value = "addressBean1")
	private Address temporaryAddress;
	@Autowired(required = false)
	@Qualifier(value = "addressBean2")
	private Address permanentAddress;


	public Person() {

	}
	public Person(Long adharCard, String name, LocalDate birthdate, Address
temporaryAddress,
			Address permanentAddress) {
		super();
		this.adharCard = adharCard;
		this.name = name;
		this.birthdate = birthdate;
		this.temporaryAddress = temporaryAddress;
		this.permanentAddress = permanentAddress;
	}


	public Person(Long adharCard, String name, String birthdate, Address
temporaryAddress,
			Address permanentAddress) {
		super();
		this.adharCard = adharCard;
		this.name = name;
		this.birthdate = LocalDate.parse(birthdate);
		this.temporaryAddress = temporaryAddress;
```

```
            this.permanentAddress = permanentAddress;
        }
        public Long getAdharCard() {
            return adharCard;
        }
        public void setAdharCard(Long adharCard) {
            this.adharCard = adharCard;
        }
        public String getName() {
            return name;
        }
        public void setName(String name) {
            this.name = name;
        }
        public LocalDate getBirthdate() {
            return birthdate;
        }
        public void setBirthdate(LocalDate birthdate) {
            this.birthdate = birthdate;
        }
        public void setCustomBirthdate(String birthdate) {
            this.birthdate = LocalDate.parse(birthdate);
        }
        public Address getTemporaryAddress() {
            return temporaryAddress;
        }
        public void setTemporaryAddress(Address temporaryAddress) {
            this.temporaryAddress = temporaryAddress;
        }
        public Address getPermanentAddress() {
            return permanentAddress;
        }
        public void setPermanentAddress(Address permanentAddress) {
            this.permanentAddress = permanentAddress;
        }
        @Override
        public String toString() {
            return "Person [adharCard=" + adharCard + ", name=" + name + ",
birthdate=" + birthdate + ", temporaryAddress="
```

```
                                + temporaryAddress + ", permanentAddress=" +
permanentAddress + "]";
        }




}

—----------------------------------------------spring3.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"
  xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="
     http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
     http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd"> <!-- bean
definitions here -->
        <context:annotation-config/>

        <bean id="addressBean1" class="com.wipro.model.Address">
                <constructor-arg name="doorNumber" value="3-4-356"/>
                <constructor-arg name="street" value="MG Street"/>
                <constructor-arg name="locality" value="Abids"/>
                <constructor-arg name="city" value="Hyderabad"/>
                <constructor-arg name="pincode" value="500001"/>
        </bean>

        <bean id="addressBean2" class="com.wipro.model.Address">
                <property name="doorNumber" value="8-6-112"/>
                <property name="street" value="Kings Street"/>
                <property name="locality" value="Bandra"/>
                <property name="city" value="Mumbai"/>
                <property name="pincode" value="400001"/>
        </bean>

        <!--  Inject the above address beans into person bean, use ref attribute -->
```

```
	<bean id="personBean" class="com.wipro.model.Person">
		<property name="adharCard" value="988612548769"/>
		<property name="name" value="Vinod Kumar"/>
		<property name="customBirthdate" value="1990-10-15"/>

	</bean>

</beans>
```

—----------

BeanFactory is the root interface while ApplicationContext is sub-interface of BeanFactory with additional features.

Implementation classes of ApplicationContext interface:

- **AnnotationConfigApplicationContext container.**
- **AnnotationConfigWebApplicationContext.**
- **XmlWebApplicationContext.**
- **FileSystemXmlApplicationContext.**
- **ClassPathXmlApplicationContext.**

—----------App.java—----------

```
package com.wipro.app;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.wipro.model.Person;
public class App {
	public static void main(String[] args) {
		ApplicationContext context=null;
		try {
			//IoC container
//			context= new
ClassPathXmlApplicationContext("spring1.xml");
//			context= new
ClassPathXmlApplicationContext("spring2.xml");
```

```
                        context= new
ClassPathXmlApplicationContext("spring3.xml");

                        Person person = (Person) context.getBean("personBean");
                        System.out.println(person);


                         //shutdown IoC

((AbstractApplicationContext)context).registerShutdownHook();
                }catch(Exception e) {
                        e.printStackTrace();
                }finally {
                        ((AbstractApplicationContext)context).close();
                }
        }
}
```

—----------

Java Configuration

Replacing xml file with Java class. The configuration metadata instead of describing in
xml file, we specify in Java class.

```
@Configuration
public class ClassName{

}
```

—----------

For the above use AnnotationConfigApplicationContext IoC container

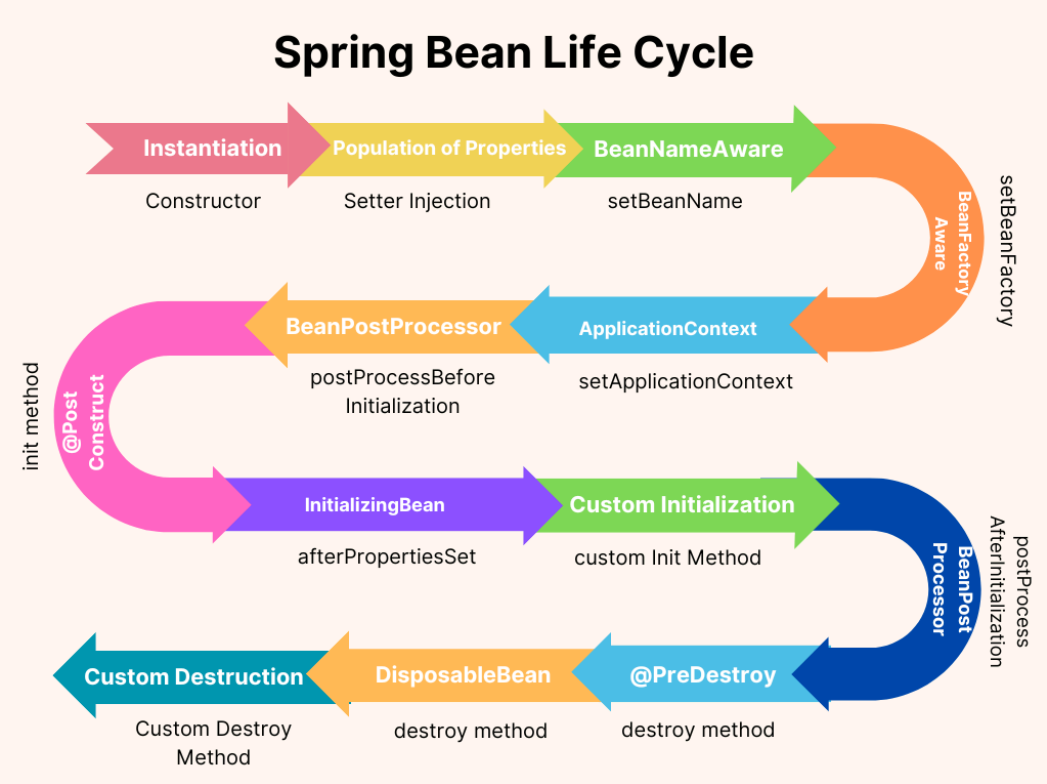
—----------SpringConfiguration.java—----------

```
package com.wipro.config;
import java.time.LocalDate;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

```
import com.wipro.model.Address;
import com.wipro.model.Person;
@Configuration
public class SpringConfiguration {
	@Bean
	public Address addressBean1() {
		return new Address("1-2-3-567","James
Street","Panty","Secunderabad", 500029L);
	}

	@Bean
	public Address addressBean2() {
		return new Address("11-12-13-1567","Queens Street","MG
Road","Secunderabad", 500039L);
	}

	@Bean
	public Person personBean() {
		return new Person(987879657424L,"Smith",LocalDate.of(1989,
10,10),addressBean1(),addressBean2());
	}
}

—----------------------------------------
package com.wipro.app;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.support.AbstractApplicationContext;
import com.wipro.config.SpringConfiguration;
import com.wipro.model.Person;
public class App1 {
	public static void main(String[] args) {
		ApplicationContext context=null;
		try {
			//IoC container
			context= new
AnnotationConfigApplicationContext(SpringConfiguration.class);

			Person person = (Person) context.getBean("personBean");
```

```
                System.out.println(person);


                //shutdown IoC

((AbstractApplicationContext)context).registerShutdownHook();
            }catch(Exception e) {
                e.printStackTrace();
            }finally {
                ((AbstractApplicationContext)context).close();
            }
        }
}
```

Bean Life Cycle



—----------

```
package com.wipro.model;
public class User {
	private String userid;
	private String password;


	public User() {

	}
	public User(String userid, String password) {
		super();
		this.userid = userid;
		this.password = password;
	}
	public String getUserid() {
		return userid;
	}
	public void setUserid(String userid) {
		this.userid = userid;
	}
	public String getPassword() {
		return password;
	}
	public void setPassword(String password) {
		this.password = password;
	}
	@Override
	public String toString() {
		return "User [userid=" + userid + ", password=" + password + "]";
	}

	public void customInitialize() {
		System.out.println("customInitialize() method executed");
	}

	public void customDestroy() {
		System.out.println("customDestroy() method is executed");
	}
```

```
	@PostConstruct
	public void postConstruct() {
		System.out.println("postConstruct() method is executed");
	}

	@PreDestroy
	public void preDestroy() {
		System.out.println("preDestroy() method is executed");
	}


}
```

—----------spring.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"
  xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="
     http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
     http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd"> <!-- bean
definitions here →

<context:annotation-config/>

	<bean id="userBean" class="com.wipro.model.User"
init-method="customInitialize" destroy-method="customDestroy">
		<property name="userid" value="admin"/>
		<property name="password" value="admin@123"/>
	</bean>


</beans>
```

—----------App.java—----------

```
package com.wipro.app;
```

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.wipro.model.User;
public class App {
	public static void main(String[] args) {
		ApplicationContext context=null;
		try {
			//IoC container
			context= new ClassPathXmlApplicationContext("spring.xml");

			User user = (User) context.getBean("userBean");

			System.out.println(user);

			 //shutdown IoC

((AbstractApplicationContext)context).registerShutdownHook();
		}catch(Exception e) {
			e.printStackTrace();
		}finally {
			((AbstractApplicationContext)context).close();
		}
	}
}
```

—----------

To use @PostConstruct and @PreDestroy annotation, specify the following dependency in pom.xml since they are from J2EE not Spring.

```
<!-- https://mvnrepository.com/artifact/jakarta.annotation/jakarta.annotation-api
-->
	<dependency>
	   <groupId>jakarta.annotation</groupId>
	   <artifactId>jakarta.annotation-api</artifactId>
	   <version>3.0.0</version>
	</dependency>
```

—----------

```
package com.wipro.model;
import org.springframework.beans.BeansException;
import org.springframework.beans.factory.config.BeanPostProcessor;
import org.springframework.stereotype.Component;
@Component
public class MyBeanProcessor implements BeanPostProcessor{
	@Override
	public Object postProcessBeforeInitialization(Object bean, String
beanName) throws BeansException {
		System.out.println("postProcessBeforeInitialization() method
executed");
		return
BeanPostProcessor.super.postProcessBeforeInitialization(bean, beanName);
	}
	@Override
	public Object postProcessAfterInitialization(Object bean, String beanName)
throws BeansException {
		System.out.println("postProcessAfterInitialization() method
executed");
		return BeanPostProcessor.super.postProcessAfterInitialization(bean,
beanName);
	}

}

—---------spring.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"
  xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="
    http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd"> <!-- bean
definitions here -->
	<context:annotation-config/>

	<context:component-scan base-package="com.wipro"/>
```

```
        <bean id="userBean" class="com.wipro.model.User"
init-method="customInitialize" destroy-method="customDestroy">
                <property name="userid" value="admin"/>
                <property name="password" value="admin@123"/>
        </bean>


</beans>
```

—------output—--------

```
postProcessBeforeInitialization() method executed
postConstruct() method is executed
customInitialize() method executed
postProcessAfterInitialization() method executed
User [userid=admin, password=admin@123]
preDestroy() method is executed
customDestroy() method is executed
```