Prasanna Wadekar
9970248407

37

SUNBEAM
Institute of Information Technology
Placement Initiative

+ **Java SE**
  - JVM (JRE) is on Desktop Machine
  - Console application, Windows application, Libraries (jars)

+ **Java Enterprise Edition**
  - Create Enterprise applications
  - Typically n-tier applications
  - Is a specification given by Sun Microsystems?
  - It is implemented by the web servers and application servers
  - Specification includes:
    - Servlet, JSP & Java Beans, JSF, EJB, JPA
  - Java EE also has many Third Party Frameworks:
    - Struts, Spring, Hibernate, etc.
  - **Web Server**
  - Application/sw that allow to execute one or more web applications in it, which can be accessed over the network (intranet or internet)
  - Java Web Server = Web Container + Extra Services
    - Web Container = Servlet Engine + JSP Engine + etc.
    - Extra Services = JNDI + Connection Pooling + etc.
  - Examples:
    - Apache Tomcat, Lotus Domino, etc.
  - **Application Server**
    - is more than a web server; serves business logic.
    - Java Application Server = Web Container + EJB Container + Extra Services
    - Examples:
    - Sun Glassfish, JBOSS, WebSphere, WebLogic, Pramati, Enhydra, Orion, etc.

  - **Apache Tomcat - Web Server**
    |- bin - scripts to start or stop web server & tomcat executables
    |- conf - contains xml files for conf web server e.g. port configuration
    |- lib - contains lot of jar files implementing java ee specs
    |- logs - for maintaining logs
    |- temp - for temp files
    |- webapps - hot deployment directory - where the web applications are deployed
    |- work - contains intermediate files created during execution of the application (useful for JSP Execution)

+ **Web Application**
  - Collection of web pages and resources.
  - Web pages can be static (HTML) or dynamic (processed at server)

+ **HTTP protocol**
  - connection-less, state-less
  - request response model

PRASANNA

- **HTTP Request**
  - From Browser to Web Server
  - http://servername:port/application/webpage
  - Request Contents:
    - Server Name (IP) & Port Num
    - Resource Identifier (/application/webpage)
    - Request Method - GET / POST
    - Request length
    - Request Headers (info about the client)
    - Cookies
    - Request Body

- **HTTP Response**
  - From Web Server to Browser
  - Response Contents:
    - Resp Status code e.g. 403, 404, 500, 200, etc.
    - Content/MIME Type e.g. text/html, text/xml, text/plain, image/png, image/jpg, image/gif, audio/mp3, video/mpeg, etc.
    - Response length
    - Response headers (info about the server)
    - Cookies
    - Response Body

- **HTTP Request Methods:**
  - **GET:**
    - data is sent via url (querystring)
    - not secure
    - send limited data
    - faster
  - **POST:**
    - data is sent via req body
    - secure
    - no data limit
    - slower
  - **HEAD:**
    - resp only contain headers not body
  - **PUT:**
    - for upload/put a file on web server
  - **DELETE:**
    - for delete a file from web server
  - **TRACE:**
    - for dubugging/tracing http req/resp data
  - **OPTIONS:**

- to know which methods are supported on server side

+ JavaEE WebApplication Directory Structure e.g. myweb:
  - [myweb]
    |- *.html, *.htm, *.jsp, ...
    |- *.jpg, *.png, ...
    |- [WEB-INF]
      |- web.xml
      |- [classes]
      | |- *.class in pkg
      |- [lib]
      | |- *.jar third party jars
      |- [src]
        |- *.java files

+ Servlet:

- Servlet is a java class, which is executed within web container when client makes request and generate response which is sent to the client.

The servlet class must be directly or indirectly inherited from Servlet interface.

- javax.servlet.Servlet interface
  + void init(ServletConfig config) throws ServletException;
    - For the first request from the client, web container load the servlet class in memory and create its object
    - Immdiately after creating the object, it call its init() method (only once in life cycle of servlet object)
    - Programmer should override this method for performing one time initialization.
    - ServletConfig object (param) is used to access servlet config info from web.xml file e.g. init params
    - If there is any exception/error during initialization, you must throw Servlet Exception.

  + **void destroy();**
    - When servlet is object is no longer used or web container is going down, then it calls destroy() method before destroying servlet object. (only once in life cycle of servlet object)
    - After call to this method, object is ready for garbage collection.
    - Programmer should override this method for performing deinitialization task.

  + void service(ServletRequest req, ServletResponse resp) throws ServletException, IOException;
    - For each request web container call the service() method.
    - Programmer should override this method to handle the request, process it and generate response.

- after method is executed, web server will send the generated resp to the client.
- the ServletRequest and ServletResponse objects are created by the web container before calling this method.

- javax.servlet.GenericServlet class
  - this class implements basic functionality of protocol-independent servlet
  - implements all methods from "Servlet" interface except service() method, hence this class is abstract class.

- javax.servlet.http.HttpServlet class
- this class implement basic functionality of the servlet which will be accessed using http protocol.
- This class has methods corresponding to each req method.
i.e. doGet(), doPost(), doHead(), doPut(), ...
- void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException;
- void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, **IOException;**
  - HttpServlet implement service() method and within that method, it checks the req method type and call corresponding doXXXX() method.

+ Servlet Example:

```
package pkg;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloServlet extends HttpServlet {
    public void init(ServletConfig config) throws ServletException {
        super.init(config);
        // ...
    }
    public void destroy() {
        // ...
    }
    public void doGet(HttpServletRequest req, HttpServletResponse resp) throws
    ServletException, IOException {
        processRequest(req,resp);
    }
    public void doPost(HttpServletRequest req, HttpServletResponse resp) throws
     ServletException, IOException {
        processRequest(req,resp);
    }
    public void processRequest(HttpServletRequest req, HttpServletResponse resp) throws
    ServletException, IOException {
```

```
                resp.setContentType("text/html");
                PrintWriter out = resp.getWriter();
                out.println("<html>\n<body>");
                out.println("<h2>Hello Servlet</h2>");
                out.println("</body>\n</html>");
        }
}
```

+ Servlet declaration in web.xml:

+ simple declaration:

```
<servlet>
        <servlet-name>HelloServlet</servlet-name>
        <servlet-class>pkg.HelloServlet</servlet-class>
</servlet>
<servlet-mapping>
        <servlet-name>HelloServlet</servlet-name>
        <url-pattern>/hello</url-pattern>
</servlet-mapping>
```

+ Alternatively, servlet can be declared using annotation @WebServlet.

```
@WebServlet("/hello")
public class HelloServlet extends HttpServlet {
        // ..
}
```

+ <load-on-startup>

- By default servlet class is loaded and instantiated, when first request is arrived for the servlet.
- Using <load-on-startup> under <servlet> tag, servlet can be loaded and instantiated as soon as application is deployed.
- e.g. <load-on-startup>1</load-on-startup>
- The number given in body of tag, indicate sequence of servlet loading, if there are multiple servlets with load-on-startup tag.

+ <init-param>

- Some configurable information can be associated with servlet using init-param under <servlet> tag in web.xml.
- Example:

```
<init-param>
        <param-name>DB_URL</param-name>
        <param-value>jdbc:mysql://localhost:3306/test</param-value>
</init-param>
```

- This value can be accessed in servlet class using ServletConfig object as:

```
ServletConfig config = this.getServletConfig();
```

```
String db_url = config.getInitParameter("DB_URL");
// ...
```

+ State Management:

+ Client Side State Management:
  - the state/info of client is stored on client machine
  - less memory is needed at the server side
  - less secure, client can access and/or modify
  - Ways: Cookie, QueryString, Hidden Form Fields

+ **Server Side State Management:**
  - the state/info of client is stored on server machine
  - large memory is needed at the server side
  - more secure, client cannot directly access
  - Ways: Session, Application

+ Cookie:

- To store info about the client, server send that info in textual key-value form to the client known as "cookie".
- Once cookie is received by the client, thereafter with each request cookie is sent back to the server.
- Cookie can store only text data upto 4KB.
- **Cookies Types:**
  - **Temporary Cookie:**
  - cookie is stored in browser memory and will be destroyed when browser is closed.
  - **Persistent Cookie:**
  - cookie is stored in client machine (disk) as text file and will be persisted even if browser is closed.

- To create and send cookie:

```
Cookie c = new Cookie("key", "value");
// c.setMaxAge(secs); // persistent
resp.addCookie(c);
```

- To receive cookie and get data:

```
Cookie[] arr = req.getCookies();
for(Cookie c : arr) {
    if(c.getName().equals("key")) {
        String value = c.getValue();
        // ...
    }
}
```

- Cookie methods:

Cookie(String name, String value);
String getName();
String getValue();

- **Session:**
  - For each client one session object is created on the server side, in which client specific state/info can be saved.
  - Session object is like a map, where data is stored in key/value pairs.
  - The key must be a string, while value can be any object.

  - **HttpSession methods:**

```
void setAttribute(String key, Object value);
Object getAttribute(String key);
void invalidate();
boolean isNew();
String getId();
```

  - **To get session object:**

```
HttpSession session = req.getSession();
```

  - to change session timeout:- web.xml in <web-app>

```
<session-config>
    <session-timeout>10</session-timeout>
</session-config>
```

  - **Session Tracking:**
    - Each session is identified using a unique session id, which is associated with the client.
    - There are two ways of this association (tracking):
      - **using sessionid cookie:**
        - by default, when new session is created (req.getSession() is called first time) a cookie is created and sessionid is sent to client via that cookie.
        - for suqsequent calls to req.getSession() access the appropriate session object by getting sessionid from that cookie.
      - **using url rewriting:**
        - In case cookies disabled, sessionid can be maintained using url.
        - resp.encodeURL() and resp.encodeRedirectURL() methods are used to embed sessionid into the url.
          e.g. http://server:port/app/page;jsessionid=374334

+ **ServletContext : (application)**
  - For each web application, web container creates a singleton object called as "ServletContext" during application deployment.
  - This object is used

- To store the data globally so that it can be accessed for all request to all pages (servlets) by all users.
- To navigate from one page (servlet) to another using RequestDispatcher.
- in servlet / config class:
  ServletContext getServletContext();
- To save and retrieve data from the ServletContext:
  void setAttribute(String key, Object value);
  Object getAttribute(String key);

+ Context Parameter:

+ ServletConfig can be used to get init param for specific servlet (from web.xml).
+ In web.xml we can declare the values which can be accessible in entire application in form of context params:

```
<web-app>
    // ...
    <context-param>
        <param-name>color</param-name>
        <param-value>yellow</param-value>
    </context-param>
    // ...
</web-app>
```

+ This param can be accessed into the web application via ServletContext object as:
  String colorValue = context.getInitParameter("color");

+ Listeners:

- These interfaces are used to handle events in the web application e.g. application start (context initialization), application stop (context uninit), session start, session end, new value added into session, any value removed from session, etc.

- interface ServletContextListener:
  void contextInitialized(ServletContextEvent e);
  void contextDestroyed(ServletContextEvent e);

- interface HttpSessionListener:
  void sessionCreated(HttpSessionEvent e);
  void sessionDestroyed(HttpSessionEvent e);

- interface HttpSessionBindingListener:

- interface HttpSessionAttributeListener:
  - void attributeAdded(HttpSessionBindingEvent e);
  - void attributeRemoved(HttpSessionBindingEvent e);
  - void attributeReplaced(HttpSessionBindingEvent e);

- **How to use listeners?**
  - step1: write a class implementing required listener
  - step2: write the appropriate logic into appropriate method
  - step3: inform web container about this listener in web.xml

- example:

```
public class Global implements ServletContextListener, HttpSessionListener {
        ServletContext ctx;
        public void contextInitialized(ServletContextEvent e){
                System.out.println("contextInitialized");
                ctx = e.getServletContext();
                ctx.setAttribte("hits", new Integer(0));
        }
        public void contextDestroyed(ServletContextEvent e){
        }
        public void sessionCreated(HttpSessionEvent e){
                System.out.println("sessionCreated");
                Integer count = (Integer)ctx.getAttribute("hits");
                count = count + 1;
                ctx.setAttribute("hits", count);
        }
        public void sessionDestroyed(HttpSessionEvent e){
        }
}
```

- **Declare listener in web.xml**

In web.xml -> <web-app>

```
<listener>
        <listener-class>sun.krd.bkshop.Global</listener-class>
</listener>
```

- Alternatively listener can be declared using annotation @WebListener

```
@WebListener
public class Global implements ServletContextListener, HttpSessionListener {
        // ...
}
```

+ Navigation:

+ Redirection:
  - resp.sendRedirect(url);
  - When sendRedirect() is called, a temp response (status code 302 and destination url) is sent to the browser; due to which browser make a new request to the new link.

- In this case two requests are originated from the browser and hence browser is aware of the navigation.
- This is slower process.
- Can redirect from any page to any other page of the same application or different application.

+ RequestDispatcher:

- RequestDispatcher rd = req.getRequestDispatcher("url");

OR RequestDispatcher rd = ctx.getRequestDispatcher("/url");

- rd.forward(req, resp);

OR rd.include(req, resp);

- Only one request is originated from the client, and single response is given back.
- This is faster.
- Browser is not aware of the navigation.
- Navigation can be done only to the pages within the same application.
- The req object can be used to carry extra info using:

```
void setAttribute(String key, Object value);
Object getAttribute(String key);
```

+ Forwarding:

- Request is forwarded to next servlet from which response will be given to the client.

+ Including:

- Request is given to next servlet, which performs some processing and return back to the calling servlet. The response generated by the second servlet will be included into first servlet's response.

+ JSP:

- Servlet = Business Logic (Java) + Presentation Logic (HTML)
- JSP = Presentation Logic (HTML) + Business Logic (Java)
- JSP => Servlet

+ Stages for compilation and execution of JSP:

---------- JSP Engine ----------

1. Translation Stage:

- When first request is made for the JSP page, it will be loaded by the web container inside JSP engine.
- JSP engine translate the JSP page into a servlet's java code. This .java file can be found in tomcat's "work" folder.
- If there is any error in JSP syntax (e.g. scriptlets), then this stage fails.

2. Compilation Stage:

- The translated servlet's .java file will be compiled into a .class file at runtime.
- If there is any java syntax error, then this stage fails.

---------- Servlet Engine ----------

3. Instantiation (Loading) Stage:

- The .class file will be loaded and object of the translated servlet will be created.
- Immediately after this init method of the JSP i.e. jspInit() will be executed.
- If this method throws any exception, this stage fails.
- This stage is also called as Loading or Initialization stage.

4. Request Handling Stage:

- All above stages are done only for the first request of the JSP file; However this stage is executed for each request.
- For each request, _jspService() method is executed (which is made up of all the scriptlet and expressions in the JSP file).

5. Destruction Stage:

- When servlet object is no longer used or web container is going down, jspDestroy() method will be executed after which servlet object will be garbage collected.

+ JSP syntax:

1. directive <%@ .... %>

- <%@page language="java" import="java.util.*, java.io.*" ..%>
  - mainly controls servlet translation
- <%@include file="file.ext" %>
  - adding external html or jsp file statically.
- <%@taglib .... %>
  - used for custom and third-party tags

2. declaration <%! .... %>

- used to declare fields and methods which will not be executed per request. e.g. jspInit(), jspDestroy(), other methods, all fields, etc.
- can write one or more declaration blocks

3. scriptlet <% .... %>

- used to write java statements to be executed per request.
- all code written here will be part of _jspService() method during translation phase.

4. expression <%= .... %>

- the java expressions whose result (string) will be directly added into generated html.

+ Implicit JSP objects:

- Few objects are directly accessible in request handling stage (without need of their creation) - in scriptlets and expressions - called as "implicit objects".

1. request: HttpServletRequest
2. response: HttpServletResponse

3. config: ServletConfig
4. application: ServletContext
5. session: HttpSession
6. out: JspWriter
7. page: Object -> this pointer
8. pageContext: PageContext -> to save some data on page scope (accessible on that page for that request only)
9. exception: Throwable -> accessible only in error pages

+ First JSP Example:

```
import java.util.*;
class FirstServlet extends HttpServlet {
        List<String> list = new ArrayList<String>();
        void init(...) ... {
                list.add("nilesh");
                list.add("rahul");
                list.add("sandeep");
                System.out.println("init() called");
        }
        void destroy() {
                System.out.println("destroy() called");
        }
        void doGet(...) {       process(...);   }
        void doPost(...) {      process(...);   }
        void process(...) {
                System.out.println("process() called");
                resp.setContentType("text/html");
                PrintWriter out = resp.getWriter();
                out.println("<html>...<body>");
                out.println("<h2>JSP Demo</h2>");
                for(String str : list)
                        out.println(str + "<br/>");
                out.println("</body>...</html>");
        }
}
```

+ @page directive:

- language="java"
- contentType="text/html"
  - will be converted to resp.setContentType("text/html");
- import="packagename"

- session="true"

- true: internally req.getSession() will be executed.
- false: session will be null.

- buffer="8kb" autoFlush="true"
- all resp generated will be stored in a temp buffer before resp is sent to the client.
- autoFlush=true : when resp is generated or buffer is full it will be automatically flushed to the client.
- autoFlush=false : the contents will be immediately flushed to the client.

- isThreadSafe="true"

- true: single thread will be used service requests. Due to which thread sync problems may not occur.
- false: for concurrent requests multiple threads will be created and multiple servlet objects will be created. This reduces performance.
- internally generated servlet implement "SingleThreadModel" marker interface, which is deprecated with newer versions.

- isErrorPage="false"

- false: simple jsp page
- true: this page is used to display some error to the end user (custom error page). This page will have access to "exception" object.

- errorPage="myerr.jsp"

- this attr is used in normal jsp page (not in err page)
- if current page get some error, web container will redirect to the given error page.

- The error pages are used to display well-formatted error messages. The error pages can be associated with some error codes in web.xml file as follows:

```
<web-app>
    // ...
    <error-page>
        <error-code>404</error-code>
        <location>/not.jsp</location>
    </error-page>
</web-app>
```

+ The JSP having minimum scriptlets is considered as better JSP. So programmers should avoid writing scriptlets wherever possible.
This can be achieved with help of using standard tags (actions), third party tags or custom tags.

+ JSP standard actions:

- JSP built-in tags used to specific tasks.

1. jsp:plugin & jsp:fallback

- these tags are used to embed applets into jsp pages.

example:

```
<jsp:plugin type="applet" code="MyApplet.class" codebase="/application" width="200" height="300">
    <jsp:fallback>Applet Not Loaded</jsp:fallback>
</jsp:plugin>
```

- This code will be internally converted into html's <object> or <embed> tag.

2. jsp:forward, jsp:include and jsp:param

```
<jsp:forward page="pageurl"/>
```

OR

```
<jsp:include page="pageurl"/>
```

- Will get converted into .....

```
RequestDispatched rd = req.getRequestDispatcher("pageurl");
rd.forward(req, resp);          OR          rd.include(req, resp);
```

* jsp:include is used to for dynamic inclusion of the file.

```
<jsp:forward page="page2.jsp">
    <jsp:param name="key1" value="value1"/>
</jsp:forward>
```

So that in next page page2.jsp,

```
String val = request.getParameter("key1");
```

3. jsp:useBean, jsp:setProperty, jsp:getProperty

+ JSP and Java Beans

+ Java Bean:

- Simple java class having fields, param less constructor, getters/setters, one or more business logic methods

```
// CapBean.java
// java bean class
public class CapBean {
    // fields
    private String word;
    // param less ctor
    public CapBean() {
        this.word = "";
    }
    // getters/setters
    public void setWord(String word) {
        this.word = word;
    }
    public String getWord() {
        return this.word;
    }
```

```
        // business logic method
        public String getCapital() {
            String res = word.toUpperCase();
            return res;
        }
    }

    // input.jsp
    <form method="post" action="out.jsp">
        Word : <input type="text" name="in_word"/>
        <input type="submit" value="Submit"/>
    </form>

    // output.jsp
    <!-- create obj of bean -->
    <jsp:useBean id="cb" class="pkg.CapBean" scope="page"/>

    <!-- call setters to set property -->
    <jsp:setProperty name="cb" property="word" param="in_word"/>

    <!-- call getters to get property -->
    Word : <jsp:getProperty name="cb" property="word"/>

    <!-- call business logic method -->
    Result : <%= cb.getCapital() %>
    Result : <jsp:getProperty name="cb" property="capital"/>
```

+ **Rules to write Java Beans:**
  - Write a simple java class having following members:
    1. one or more fields
    2. param less constructor
    3. getter/setter methods (proper camel case)
    4. one or more business logic method (any name)

+ Standard actions for java beans:
  1. <jsp:useBean id="obj" class="pkg.ClassName" scope="page"/>
  - This tag check the whether any object is present in the given scope with given name (id).
    - If object is present, it will be accessed by that name.
    - If object is not present, class will be loaded and object will be created at runtime.
    - Then object will added into given scope with given name.

- **Java Bean scopes:**
  i. application: bean is stored into ServletContext and hence accessible in all requests to all pages by all users

ii. session: bean is stored into HttpSession and hence accessible in all requests to all pages by current user.

iii. request: bean is stored into HttpServletRequest and hence accessible in all the pages on which request is forwarded or included.

iv. page: bean is stored into PageContext and hence accessible in current request to current page.

2. **<jsp:setProperty name="obj" property="propName" value="fix_value"/>**
<jsp:setProperty name="obj" property="propName" param="req_param"/>
- Internally find corresponding setter method (i.e. setPropName()) and execute it on given object.
- "value" is used to give fix value, while "param" is used to give value from request parameter.

3. **<jsp:getProperty name="obj" property="propName"/>**
- Internally find corresponding getter method (i.e. getPropName()) and execute it. The return value will be added into html response.

* Before calling <jsp:setProperty> or <jsp:getProperty> the jsp page must have <jsp:useBean> tag.

+ Model I architecture:
- Also called as "Model-View" Architecture.
- View -> Presentation Logic -> JSP pages
- Model -> Business Logic -> Java Beans
- used for very small web applications

+ Model II architecture:
- Also called as "Model-View-Controller" Architecture.
- e.g. Third party frameworks like struts, spring, etc.

+ JSP Expression Language:
- Used to access (read) values from different scopes without using scriptlet or expression syntax.
- EL syntax:
  ${scopeName.variableName}
- Four scopes:
  - pageScope, requestScope, sessionScope, applicationScope
  e.g. ${sessionScope.myname}
- This syntax can be further simplified by omitting scopeName.
  i.e. ${varName}

In this case, the var will be searched from the lowest scope to highest scope.
- Using this you can call getter methods of the object as follows:
  ${scopeName.objName.propName}

e.g.${sessionScope.lb.username}
- Using EL syntax you can also call any method on the object
  ${scopeName.objName.method()}
- If any EL expression is not found, it will be ignored (no error will be raised).
- You can write math expressions (of consts) into EL
  ${3 + 4 / 2 * 7}

- EL can be used to access values from the special implicit variables (objects).

EL implicit objects:

1. param
   - ${param.varName}
     -> req.getParamter("varName");
2. paramValues
   - ${paramValues.varName}
     -> req.getParamterValues("varName");
3. header
   - ${header.varName}
     -> req.getHeader("varName");
4. headerValues
5. initParam
   - ${initParam.varName}
     -> config.getInitParameter("varName");
6. cookies
   - ${cookies.cookieName}
     -> will get the value of the cookie with cookieName
7. pageContext

**** All standard actions and EL syntax must be used outside scriptlets <% ... %>**

*** All EL in the page can be ignored using
<%@ page isELIgnored="true" %>

+ JSP Tags - business logic embedded into presentation logic

+ Standard Actions:
  - Built-in tags with prefix "jsp:"
  - e.g. forward, include, plugin, fallback, beans related
+ **Third Party Tag Libraries:**
  - Struts Tag Library, Spring Tag Library, JSTL, etc.
  - JSTL - JSP Standard Tag Library - Sun Microsystems
    - Basic prog constructs: switch, if-else, loops, etc.
    - JDBC tags, etc.
+ **Custom Tags:**
  - Programmer can define his own tags for special needs

+ JSTL:

- To use JSTL in the JSP web page:

- step1: add appropriate jars into WEB-INF/lib directory
- standard.jar, jstl.jar

- step2: at the start of the JSP page use "taglib" directive
- <%@ taglib prefix="c" uri="http://java.sun.com/jstl/core_rt" %>
  * prefix = used to avoid tag name clashes

* uri = identifier for tags collection

- **step3: use the apprpriate tag with proper attributes into the JSP page. example:**

books.jsp

```
<c:forEach var="b" items="${bb.bookList}">
        <input type="checkbox" name="book" value="${b.bookid}"> ${b.name}
</c:forEach>
```

validate.jsp

```
<c:choose>
        <c:when test="${lb.status=='true'}">
                //...
        </c:when>
        <c:otherwise>
                //...
        </c:otherwise>
</c:choose>
```

+ Custom Tags:

+ JSP has mainly two types of tags:
  - classic tags and simple tags
  - both are inherited from marker interface "JspTag".
  - To implement these tags you can use interfaces "Tag" and "SimpleTag" or adapter classes "TagSupport" and "SimpleTagSupport" respectively.

+ Steps to implement custom (Simple) tag:

- step 0: decide the application, name, attributes and body of the tag.
  e.g. <my:wish uname="some_name"/>
  This tag should print greeting meesage for the given name.

- **step 1:** write the tag handler class inherited from "SimpleTagSupport" e.g. WishTag. Also add param less ctor.
  - in the class write number of fields equal to number of attributes, with names matching to attribute. Also add getter/setter for them.
  - override doTag() method of the "SimpleTagSupport" class and implement the business + presentation logic in it.

```
public class WishTag extends SimpleTagSupport {
        private String uname;
        public WishTag() {
                this.uname = "";
        }
```

```
        public void setUname(String uname) {
                this.uname = uname;
        }
        public String getUname() {
                return this.uname;
        }
        public void doTag() {
                JspContext ctx = this.getJspContext();
                JspWriter out = ctx.getOut();
                out.println("Hello, " + uname);
        }
}
```

- step 2:

- write tag library descriptor (tld) xml file inside WEB-INF to give complete information about the tag.

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-
jsptaglibrary_2_1.xsd">
    <tlib-version>1.0</tlib-version>
    <short-name>custom</short-name>
    <uri>/WEB-INF/custtags.tld</uri>
    <tag>
        <name>wish</name>
        <tag-class>sub.krd.bkshop.WishTag</tag-class>
        <body-content>empty</body-content>
        <attribute>
            <name>uname</name>
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
            <type>java.lang.String</type>
        </attribute>
    </tag>
</taglib>
```

- step3: use the tag into the JSP page
  - <%@taglib prefix="my" uri="/WEB-INF/custtags.tld" %>
  - use in page

```
<my:wish uname="${lb.username}"/>
```

+ SimpleTag life cycle:

1. When page containing custom tag is accessed first time, during translation stage:
   - the tld file is referred (via given prefix in @taglib)
   - from tld used tag is found and validated (syntax)
2. For each time tag is used, tag-class (given in .tld file) is loaded and object is created at runtime. Paramless constructor will be called here.
3. setJspContext() method will be called by the container and current JSP's PageContext object will be passed into it. This object contains all info needed to process JSP page.
4. If tag is child of any other tag, then setParent() method will be called.
5. Then container calls setter methods for all attributes used in the JSP file during tag invocation.
6. If tag has some body, then setJspBody() method will be called to provide tag body.
7. Finally container calls doTag() method of the tag class, which does server side processing and generate html output if any.
8. After doTag() is completed, the tag's generated html will be added into page response.

* If not overridden, setJspContext() method of the "SimpleTagSupport" will be called (step 3), which will save the current JspContext so that it can be accessed via call to getJspContext() [usually in doTag() method].

** Refer docs of "interface Tag" for classic tag lifecycle.

+ **Hibernate:**
- Hibernate is an ORM tool.
- ORM - Object Relational Mapping
- Java Object <= mapping => Relational Database
  - Java Class (Entity) - Database Table
  - Class Fields - Table Columns
- **SessionFactory - hibernate.cfg.xml (src directory)**
  - contains full info about the db to be connected.
  - also contains info about entity classes
  - In an application there must be single object of SessionFactory
- **Session -**
  - Encapsulate JDBC connection
  - helps transaction management
  - track the changes into entity objects and update in db
  - created from SessionFactory

* SessionFactory can get database connection by using DriverManager (same as JDBC), DataSource or any other way.

+ **Steps to use hibernate into the project:**
- step1: copy all hibernate jar files into WEB-INF/lib folder along with JDBC driver jar.
- step2: create a new xml file into src directory i.e. hibernate.cfg.xml
  - This file contains info about database or data-source

- This file also contains info about all entity classes

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
 <session-factory>
  <property name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>
  <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
  <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/dacdb</property>
  <property name="hibernate.connection.username">nilesh</property>
  <property name="hibernate.connection.password">nilesh</property>
  <property name="hibernate.connection.pool_size">2</property>
  <property name="hibernate.current_session_context_class">thread</property>
  <property name="hibernate.connection.autocommit">true</property>
  <property name="hibernate.show_sql">true</property>
  <mapping class="sun.krd.hbshop.Login"/>
 </session-factory>
</hibernate-configuration>
```

- **step3**: create an helper class, say HbUtil, to initialize hibernate SessionFactory object.

```
public class HbUtil {
	public static SessionFactory factory;
	static {
		Configuration configuration = new Configuration().configure();
		StandardServiceRegistryBuilder builder = new StandardServiceRegistryBuilder(). ,
		applySettings(configuration.getProperties());
		factory = configuration.buildSessionFactory(builder.build());
		System.out.println("Hibernate Session Factory created ...");
	}
}
```

- step4: create one or more entity classes using annotations from javax.persistence package. Also make their entries into hibernate.cfg.xml.

```
		@Entity
		@Table(name="BOOKS")
		public class Book implements Serializable {
			@Id
			@Column(name="BOOKID")
			private int id;
			private String name;
			private String author;
			private String subject;
```

```
    private double price;
    // paramless constructor
    // getters/setters
}
```

- step5: To perform db operation into servlet, JSP, DAO or any other class, create object of hibernate "Session" and call its methods. Finally close the session.

+ Hibernate Db Operations:

1. Insert Operation - Book b:

```
Session hbSession = HbUtil.factory.openSession();
hbSession.save(b);              // persist(b);
hbSession.flush();
hbSession.close();
```

2. Delete Operation - id=23:

```
Book b = new Book();
b.setBookid(id);
Session hbSession = HbUtil.factory.openSession();
hbSession.delete(b);
hbSession.flush();
hbSession.close();
```

3. Update Operation - [name,subject,price,author -> id]

```
Book b = new Book(id,name,author,subject,price);
Session hbSession = HbUtil.factory.openSession();
hbSession.update(b);
hbSession.flush();
hbSession.close();
```

4. Select by PK Operation - id

```
Session hbSession = HbUtil.factory.openSession();
Book b = hbSession.get(Book.class, id);
hbSession.close();
```

+ Hibernate Entity Life Cycle:

1. Transient:

- The entity objects created in java application and hibernate is not aware of them.

2. Persistent:

- The objects which are part of hibernate cache are in "persistent" state; where any changes in those objects will be automatically reflected into db.
- When any transient/detached objects are added into cache using save() or update() method, they become persistent.
- Also all objects created by session via get() or list() method are persistent.

3. Detached:

- Any object removed from the hibernate cache is in this state
- The evict() operation will detach a particular object; while clear() will detach all objects in the cache.

4. Removed:

- If db row corresponding to any object is deleted then object is in this state.
- Any persistent object doing delete() operation will come to this state.

+ Hibernate Relations:

- Hibernate allows association among hibernate entities as follows:
  - One to One
  - One to Many
  - Many to One
  - Many to Many
- It facilitates fetching related objects from the database easily.
- Example of One to Many relationship:

```
// Dept.java
@Entity
@Table(name="DEPT")
public class Dept {
	@Id
	private int deptno;
	private String dname;
	private String loc;

	@OneToMany(fetch=FetchType.LAZY, mappedBy="deptno")
	private List<Emp> empList;

	// constructor and getter/setters
}

// Emp.java
@Entity
@Table(name="EMP")
public class Emp {
	private int empno;
	private String name;
	private double salary;
	private int deptno;

	// constructor and getter/setters
}

// Main.java
```

```
public static void main(String[] args) {
        Session session = HbUtil.getFactory().openSession();
        Dept d =  (Dept) session.get(Dept.class, 1);
        System.out.println("DEPT : " + d);
        List<Emp> list = d.getEmpList();
        for (Emp e : list) {
                System.out.println(e);
        }
        session.close();
}
```

- Hibernate relations support two types of fetching:

1. Lazy Fetching:

- Related objects (child table data) are fetched only when they need access.
- Typically fires two different queries. First on master table while fetching its data and second on child table when element is accessed.
- In above example, session.get(Dept.class, 1) - fires:
  select DEPTNO, DNAME, LOC from DEPT where DEPTNO=?
- And, d.getEmpList() fires:
  select EMPNO, ENAME, SALARY, DEPTNO from EMP where DEPTNO=?

2. Eager Fetching:

- Related objects (child table data) are fetched along with the main object (master table data).

- Internally use single join query to fetch the results from both tables.
- In above example, if fetch type is changed to FetchType.EAGER then, session.get(Dept.class, 1) - fires:

```
select d.DEPTNO, d.LOC, d.DNAME, e.DEPTNO, e.EMPNO, e.ENAME, e.SALARY from DEPT d left
outer join EMP e on d.DEPTNO=d.DEPTNO where d.DEPTNO=?
```

+ Struts2:

+ Steps for First Struts Application:

1. Create "Dynamic Web Application" - mystruts
2. Add struts 2 jar files into WEB-INF/lib folder
3. In "src", create "struts.xml" file (with proper grammer).
4. In web.xml, make entry of the struts2 controller filter using <filter> element.
5. create "in.jsp" -> form action="caps" - textfield(intext), submit - using struts (third party) tags
6. create "CapAction" (model) class extends ActionSupport
   fields: intext, outtext; constructor; getter/setter;
   execute() method - business logic (return a string "res")
7. create "out.jsp" -> display intext & outtext using struts tags
8. modify struts.xml
   <action name="caps" class="pkg.CapAction">

```
                <result name="res">/out.jsp</result>
            </action>
```

+ Model II Architecture:

- Model View Controller architecture
- Struts is one of the implementation of MVC arch.

+ Struts2 MVC:

+ View
  - Presentation Logic
  - Implemented as JSP pages
  - JSP pages will contain many struts tags e.g. <s:form/>

+ **Model**
  - Business Logic + Database Connectivity
  - Implemented as Java classes called as "Actions"
  - Typical action class contains
    - fields [to take input from prev page and to give output to next page]
    - param less constructor, getters/setters
    - Business Logic method (mostly execute())
  - Mostly this class is inherited from ActionSupport

+ **Controller**
  - Controls all navigations (view-model, model-view, view-view, model-model)
  - Implemented as Filter i.e. org.apache.struts2.dispatcher.FilterDispatcher
  - Information about all navigations is maintained in "struts.xml" in "src" directory.
  - Each user request first hit to the controller; then controller read struts.xml to find corresponding action class or jsp page and forward request to them.

+ Model(Action)-Model(Action) Navigation:

Example: LoginAction -> SubjectAction

```
<action name="login" class="pkg.LoginAction">
    <result type="redirect-action" name="login_success">/subjects</result>
    <result name="login_fail">/error.jsp</result>
</action>
<action name="subjects" class="pkg.SubjectAction">
    ...
</action>
```

+ View-View Navigation:

Example: error.jsp -> login.jsp

in error.jsp

```
---------
<s:a href="index.action">Login Again</s:a>
```

in struts.xml

```
<action name="index">
        <result>/login.jsp</result>
</action>
```

+ To access HttpSession, HttpServletRequest, HttpServletResponse or ServletContext into the action class:

- There are static methods into ServletActionContext class:

```
HttpServletRequest req = ServletActionContext.getRequest();
HttpServletResponse resp=ServletActionContext.getResponse();
ServletContext application=ServletActionContext.getServletContext();
HttpSession session = ServletActionContext.getRequest().getSession();
```

+ **.....Aware interfaces:**

- Another efficient way of accessing request, session or application attributes into the action is implementing RequestAware, SessionAware or ApplicationAware interface.
- Each of this inteface provide a setter method with a "Map" arg. This arg contains the attributes in form of java.util.Map from which they can get() or put().
- For Example, to access session attributes in the action class:

```
class MyAction extends ActionSupport implements SessionAware {
        // other fields, constructor, getters/setters

        private Map session;
        public void setSession(Map session) {
                this.session = session;
        }

        public void execute() {
                // ...
                // to add any value in session
                session.put("key1", value1);
                // ...
                // to retrieve any value from session
                value2 = (casting)session.get("key2");
        }
}
```

+ OGNL - Object Graph Navigation Language

- Used to access variables in different scopes (like EL)
- Used with struts tags - attribute values - in JSP pages
- Can access all values on Context Map:

+ context map
|- application
|- session
|- value stack (root)
|- request
|- parameters
|- attr

+ To access variables (attributes) from application, session or request scope use syntax:
#application.varname, #session.varname, #request.varname

+ To access request parameters:
#parameters.varname

+ To access variables (attributes) in any scope:
#attr.varname
This will search the varname into page scope to application scope.

+ All the values on value stack can be accessed without any marker '#'. The object of the current action is on the top of value stack (known as root) and hence all its data members can be accessed directly without using '#' marker.
Example:
<s:property value="outtext"/> <- out.jsp
- outtext is field of CapAction.

+ #session.varname OR #session['varname']

+ Struts Tags:

```
<%-- <s:radio name="subject" list="subList"/> --%>
<s:iterator value="subList">
    <s:radio name="subject" list="{top}"/>
</s:iterator>

<%-- <s:checkboxlist name="book" list="bkList" listKey="bookid" listValue="name"/> --%>
<s:iterator value="bkList">
    <s:checkboklist name="book" list="{top}" listKey="bookid" listValue="name"/>
</s:iterator>
```

+ OGNL to EL:

- struts tags understand OGNL only (not EL).
- JSP tags, JSTL tags, Custom tags understand EL only (not OGNL).
- Sometimes there might be need of accessing OGNL values into JSP/Custom tags.
- To do this, we need to add those values into some scope using <s:set/> so that it can be accessed using EL syntax.

<s:set name="varname" value="OGNL_expr" scope="page|request|session|application"/>

* Any struts tag cannot be nested (in attr value) into another struts/custom tag.

+ **Validation:**

- Action class can contain data validation logic (at server side).
- This validation can be performed into overridden validate() method of ActionSupport.
- In case any validation error, you can use addFieldError() or addActionError() method of the **ActionSupport.**
- The field level errors will be displayed on the input page nearby respective fields.
- The action errors will be displayed on the input page using <s:actionerror/> tag.
- In this case result with name "input" must be specified under that acion into struts.xml. For example:

```
<result name="input">/login.jsp</result>
```

- If any error is added into validate() method, execute() method will not be executed.
- Also some informative messages can be displayed on the JSP pages using <s:actionmessage/>. These messages must be added by the action using addActionMessage() of ActionSupport.
  * Messages are NOT Errors.

+ **Action Business Logic:**

- By default execute() method contains business logic od the action.
- For sake of readability you may choose different name of the business logic method e.g. SubjectAction business logic may be written in a method with name fetchSubjects().

- In this case controller must be aware of the business logic method name, which can be done via struts.xml file:

```
<action name="subjects" class="pkg.SubjectAction" method="fetchSubjects">
    // ...
</action>
```

- login.jsp [user,pass]
  - LoginAction - checkLogin()
  - <action name="login" class="p.LoginAction" method="checkLogin">
- register.jsp[user,pass,cpass]
  - LoginAction - registerUser()
  - <action name="register" class="p.LoginAction" method="registerUser">
- chpass.jsp [user,pass,cpass]
  - LoginAction - changePassword()
  - <action name="chpass" class="p.LoginAction" method="changePassword">

+ **ModelDriven interface:**

- Typically Model is business object carrying data between action and view. It is a simple POJO class. Many times it represents data fetched from database (hibernate entity).

- Useful to avoid data redundancy.
- Example:

```
// Model class
class Book {
        // fields, constructor, getter/setters
}

// Action class
class AddBookAction implements ModelDriven<Book> {
        private Book book = new Book();
        public Book getModel() {
                return book;
        }
        public String execute() {
                // try-catch avoided for simplicity
                BookDao dao = new BookDao();
                dao.addBook(book);
                dao.close();
                return "success";
        }
}
```

- If action class is inherited from ModelDriven, controller invokes its getModel() method. Then data from html form is filled into the returned (from getModel()) model object. This object can be furthr used in business logic method.

+ Using Tiles with Struts 2:

1) create dynamic web project and add struts & tiles jar files into its WEB-INF/lib directory. Also add its "web.xml" file into that project.

2) create "user_master.jsp", which contains the basic html view that you want to apply for set of web pages. The configurable parts will be identified by some names like title, header, footer, body, etc using <tiles:insertAttribute> tag.

3) create default pages for above configurable parts i.e. header.jsp, footer.jsp, body.jsp, etc.

4) create the web pages, whose look and feel will be decided by base layout. i.e. subjects.jsp, books.jsp, showcart.jsp, etc.

5) create tile configuration file "tiles.xml" into WEB-INF directory.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE tiles-definitions PUBLIC
        "-//Apache Software Foundation//DTD Tiles Configuration 2.0//EN"
        "http://tiles.apache.org/dtds/tiles-config_2_0.dtd">
```

```
<tiles-definitions>
        // ...
</tiles-definitions>
```

This file contains basic html view config as :

```
<definition name="user_master" template="/user_master.jsp">
        <put-attribute name="title"  value=""/>
        <put-attribute name="header" value="/header.jsp"/>
        <put-attribute name="body"   value=""/>
        <put-attribute name="footer"   value="/footer.jsp"/>
</definition>
```

This file further contains one entry for each web page, which is extended from above "user_master" entry and override certain attributes like "title" and "body" as required. For Example:

```
<definition name="show_subjects" extends="user_master">
        <put-attribute name="title"  value="Subjects"/>
        <put-attribute name="body"   value="/subjects.jsp"/>
</definition>
```

6) Inside web.xml add context-param that informs about path of tiles.xml and also one TilesListener entry.

```
<context-param>
        <param-
name>org.apache.tiles.impl.BasicTilesContainer.DEFINITIONS_CONFIG</param-name>
        <param-value>/WEB-INF/tiles.xml</param-value>
</context-param>
<listener>
        <listener-class>org.apache.struts2.tiles.StrutsTilesListener</listener-class>
</listener>
```

7) finally go to struts.xml and declare a result type for "tiles".

```
<result-types>
        <result-type name="tiles" class="org.apache.struts2.views.tiles.TilesResult" />
</result-types>
```

8) In this file, modify the results for each web page, which will redirect you to the corresponding tile entry into "tiles.xml"

```
<action name="subjects" class="pkg.SubjectAction">
        <result type="tiles" name="show_sub">show_subjects</result>
</action>
```

+ Interceptors:

- Interceptors are java classes implementing "Interceptor" interface of struts2.
- Interceptor interface has 3 methods:
    - init()
        - immediately after creating object of interceptor
    - destroy()
        - before destroying object of interceptor
    - intercept()
    - code of this method can be splitted in 3 parts:
        - pre-processing: this part is executed before executing next element in the chain
        - invoke: invoke next element in chain and collect result
        - post-processing: this part is executed after executing next element in the chain
- To **write an interceptor:**
    step1: write an class inherited from Interceptor interface.
    step2: in intercept() method:
        - do pre-processing
        - invoke next ele in chain
        - do post-processing
    step3: make entry of the ineterceptor into struts.xml
    step4: add your interceptor into interceptor stack at proper place (create a modified interceptor stack) in struts.xml
    step5: apply interceptor stack to the intended action class in struts.xml

+ struts packages:

1. in struts.xml <package> is used to specify scope for global-results, default-interceptor-ref, etc.
    - So global-result and default-interceptor-ref are applicable only for the current package.

2. In package we can specify "namespace". This namespace is logical and will appear into the URL while accessing any "action".

LoginAction
    |- <result name="sucess" type="redirect-action">customer/sub</result>

3. All JSP pages can be secured by putting them within WEB-INF; so that these pages are not directly accessible.
    - Further these pages can be stored under different folder so that maintaining web-site become easier. In that case all URLs should be modified accordingly.
    <result name="success">/WEB-INF/customer/subjects.jsp</result>

4. struts.xml can be divided into multiple xml files for sake of better maintenance.
    - In that case, XML files should be added in main struts.xml file.
    <include file="customer-struts.xml"/>

+ Struts Architecture:
    https://struts.apache.org/docs/the-struts-2-request-flow.data/Struts2-Architecture.png

+ **Spring Framework:**

+ **Introduction:**

- Founder : Rod Johnson @ Year 2003
- Open Source Framework (Project) hosted on sourceforge.net
- Spring in Light Weight Comprehensive Framework for developing JavaSE and JavaEE application.
    - Light Weight
    - Comprehensive

+ **Why Spring?**

- Simplify Java Development
- Design with Interfaces
- Test Driven Application
- Declarative AOP (XML or Annotations)
- Easy coupling via Dependency Injection
    - Done by Spring IoC container [Inversion of Control]
    - Programmer declare fields and Spring Container initialize them via DI.
- Convert all checked exceptions into unchecked.
- Well designed - Easily Extendable - Reusable
- Extremely Modular - Flexible - Easily integrable with any existing technology

+ **Spring Architecture:**

http://docs.spring.io/spring/docs/current/spring-framework-reference/html/images/spring-overview.png

+ **Java SE Application With Spring:**

- step1: create new "Java Project".
- step2: create a "lib" folder in the project and copy required spring jars into it.
- step3: add all the jars into the classpath of the project.

- step4: in "src" directory create "Spring Bean Configuration File" with any name e.g. spring1.xml
- step5: create interface IPerson with getter/setter declarations.
- step6: implement interface into class Person
- step7: add "Person" entry into xml file and set some props

```
<bean name="p1" class="sun.krd.spring.basic01.Person">
    <property name="name" value="Nilesh"/>
    <property name="address" value="Pune"/>
    <property name="age" value="28"/>
</bean>
```

- step8: write a main class, in which create "XmlBeanFactory" object and retrieve "Person" object through its "getBean()" method. Finally call the "Person" methods from interface reference.

```
Resource res = new ClassPathResource("spring1.xml");
XmlBeanFactory factory = new XmlBeanFactory(res);

IPerson p = (IPerson) factory.getBean("p1");
System.out.println("Name : " + p.getName());
System.out.println("Addr : " + p.getAddress());
System.out.println("Age : " + p.getAge());
```

+ Dependency Injection:

- In above example, the object of Person class is created by the spring container (during call to getBean()).
- During creation by default param less constructor is called, and then container call the setter methods as given by <property ... /> tags in xml.
- This way of injecting values of data members is referred as "setter based" DI.

- If needed, values of data members can be passed via parameterized constructor.
- To do this spring xml configuration should be done as :

```
<bean name="p2" class="sun.krd.spring.basic01.Person">
        <constructor-arg index="0" value="Raj" />
        <constructor-arg index="1" value="Kaard" />
        <constructor-arg index="2" value="24" />
</bean>
```

- This way of injecting values of data members is referred as "constructor based" DI.

+ init-method and destroy-method:

- It is also possible to call some method after creation of the object (after constructor execution and DI) by specifying method name into xml file.
- Also it is possible to call some method before destruction of the object.
- These method typically contains additional initialization and de-initialization logic.
- Example:

```
class Person {
        // ...
        public void initPerson() {
                // ...
        }
        // ...
}

<bean name="p3" class="pkg.Person" init-method="initPerson">
        // ...
</bean>
```

* Alternatively (instead of declaring in xml file), these methods may have @PostConstruct and @PreDestroy annotations.

+ **XmlBeanFactory:**
- When XmlBeanFactory is created, it loads xml file and check its syntax.
- It is used to get the bean reference (by name or by type).
- When getBean() is called, factory checks whether bean object is already created. If object is already created, it simply returns reference to that object.
- If object is not created, factory will load the given class, create the object, performs DI and call init method if given.

+ **ApplicationContext:**
- When ApplicationContext is created, it will load and validate spring xml file.
- Also during creation, it will create all bean objects, performs DI and call init method if given.
- When getBean() method is called, reference of the bean will be returned.
- ApplicationContext provides all functionalities of XmlBeanFactory and also provides few others:
  - event handling & publishing
  - internatialization
  - resource management and access
  - life cycle interfaces
- two types:
  - ClassPathXmlApplicationContext
    - used with xml config of spring
    - To create context object

```
ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext("spring1.xml");
ctx.registerShutdownHook();
```

    - call to registerShutdownHook() ensures that all bean objects will be destroyed when spring container will be closed.

  - AnnotationConfigApplicationContext
    - used with annotation config of spring
    - to perform annotation config (no xml) write a seperate class (other than interface and bean class)

```
@Configuration
public class SpringBeanConfig {
    @Bean
    public Object p1() {
        Person p = new Person();
        p.setName("ABC");
        p.setAddress("XYZ");
```

```
                        p.setAge(123);
                        return p;
                    }
                }
                - in main()
                    AnnotationConfigApplicationContext ctx = new
AnnotationConfigApplicationContext(SpringBeanConfig.class);
                    ctx.registerShutdownHook();

                    IPerson p = (IPerson) ctx.getBean("p1");
```

+ Dependency and Dependent Beans:

1. interface ILog
   - void write(String str);
2. class ConsoleLog implements ILog
3. class FileLog implements ILog
4. interface IPerson
   - getters/setters for name, address, age and logger.
5. class Person implements IPerson
   - Fields: name, address, age
     private ILog logger;
   - implement all methods of IPerson
     - in getter/setter of name, address and age
       - logger.write("method() called");
6. spring xml file

```
<!-- dependency declaraions --!>
    <bean name="consLog" class="pkg.ConsoleLog">
    </bean>
    ... FileLog

<!-- dependent declaraions --!>
    <bean name="p1" class="pkg.Person">
        <setproperty name="logger" ref="consLog"/>
        .... other setters
    </bean>
```

7. main()
   use ClassPathXmlApplicationContext to get Person bean and display its details.

+ Spring Config:
  1. XML config - Only XML (no anns)
  2. Annotation config - Only Anns (no XML)
  3. Java Config - XML + Annotations

+ **Spring XML file:**
  - beans (3.0), context (3.0)
  - <context:annotation-config/>
    - Spring container will recognize annotations like @PostConstruct, @PreDestroy, @Autowired, @Qualifier, etc. on all beans.

+ **@Autowired**
  - Used on the fields of the bean, so that container will perform DI for those fields.
  - When @Autowired is used on any field, by default container will search all the beans of matching type (same type or derived type) and if single bean is found then it is injected (attached to the field).
  - If more than one matching bean is found, then autowiring fail and exception is raised.

+ **@Qualifier**
  - Mainly used to resolve the ambiguity raised during autowiring, by specifying name of the dependency bean.
  -

```
@Autowired
@Qualifier("consLog")
private ILog logger;
```

+ stereo-type annotations:
  - These annotations are used on class level, so that those classes will be treated as spring beans by the container.
  - To search all the classes with stereo type anns in certain package use

```
<context:component-scan base-package="pkgName">
</context:component-scan>
```

  - there are four stereo type annotations:

1. @Component
  - All other bean classes like simple POJOs should be annotated with @Component.

2. @Service
  - Bean classes containing business logics should be annotated with @Service. These classes are typically used business logic layer. Many times these classes internally access repositories.

3. @Repository
  - Bean classes doing data handling, database connections (in which data can be read or written) should be annotated with @Repository. These classes are typically used in backend (persistence) layer.

4. @Controller
  - Bean classes performing navigation, model-view binding should be annotated with @Controller.

+ **Spring Bean scopes:**

1. singleton:

- default scope
- when application context is loaded, one bean object is created
- each call to getBean() returns same object

2. prototype:
   - when application ctx is loaded, no bean object is created.
   - each call to getBean(Class cls) returns a new object.
3. request:
   - scope is limited to current request
   - used only in web application
4. session:
   - scope is limited to current session
   - used only in web application

* Spring bean scopes can be declared into spring xml file <bean /> tag or @Scope anns on the bean class.

```
@Component
@Scope("prototype")
class Book {
	// ...
}
```

+ Using Stereotype annotations for implementing simple assign:

```
- @Component
@Scope("prototype")
class Book {
	// ...
}
- @Repository("bkDao")
class BookDao {
	private List<Book> bkList;
	// ctor, getter/setter
	// insert(), delete(), search(), ...
}
- @Service
class BookService {
	@Autowired
	private BookDao dao;
	// ctor, getter/setter
	// addBook(), delBook(), findBook(), getBookPrice() ...
}
- class BookMain {
	main() {
		// menu driven program
```

```
                    // BookService obj -> ctx.getBean(BookService.class);
                    // Book obj -> ctx.getBean(Book.class);
                }
        }
        - spring xml file
        <context:component-scan base-package="sun.krd.spring.shop"/>
        <bean name="bkDao" class="sun.krd.spring.shop.BookDao">
                <property name="bkList">
                        <list>
                                <bean class="sun.krd.spring.shop.Book">
                                        <property name="name" value="book1"/>
                                        <property name="price" value="11"/>
                                </bean>
                                ...
                        </list>
                </property>
        </bean>

+ SPEL : Spring Expression Language
        - Can be used in JSP pages and also in spring bean classes using @Value annotation.
        - syntax: #{var.member}, #{var.method()}
        - in bean classes @Value annotation is used for DI (instead of @Autowired and @Qualifier)
        Example:
                class BookService {
                        @Value("#{bkDao}")
                        private BookDao dao;
                        // ...
                }
+ Assign:
        class AccHolder {
                String name, address, mobile;
                // ...
        }
        class Account {
                int accid;
                String acctype;
                double balance;
                * use SPEL
                AccHolder accHolder;
                // ...
                // withdraw(), deposit()
        }
        class AccountMain {
```

```
main() {
    // create Account Object
    // get its data from user
    // perform withdraw(), deposit()
    // print balance after each operation
}
}
```

+ **Aspect Oriented Programming:**
  - The business logic is seperated from cross-cutting concerns
  - The cross-cutting concerns are different aspects other than actual business logic e.g. logging, monitoring, security, etc
  - Due to this code will be modularized and business logic, other concerns will be implemented seperately (by different developers).

+ AOP Terminologies:
  - Aspect:
    - Additional fuctionality (cross cutting concern) other than business logic
    - In spring, implemented as a seperate class annotated with @Aspect
  - Advice:
    - The operation to be performed to handle certain cross-cutting concern
    - In spring it is implemented as methods in aspect class
    - There are four types of advices given by different annotations (on advice methods):
      - @Before - pre-processing
      - @After - post-processing
      - @Around - pre-processing and post-processing
      - @AfterThrowing - post-processing in case business logic method throws exception
  - JoinPoint:
    - represent the point in the application where advice is to be applied.
    - In spring it is an "JoinPoint" object which contains information of the business logic on which advice is applied. It also contains information about the object on which business logic method is called.
  - **Target:**
    - The object on which business logic method is called.
  - Pointcut:
    - combination of one or more join points where advice is to be applied
    - In spring PointCut is implemented as an empty method with @PointCut annotation
  - Proxy:
    - This object is internally created by the spring container to call the advices and the actual business logic.
  - Advisor:
    - group of advice and pointcut into a single unit called as "Advisor".
    - This object is internally passed to proxy factory to create the proxy object.

+ AOP Example:

- create new java project
- create a "lib" folder and add spring jars as well as aop jars.
- add these jars into the classpath of the project.

- implement interfaces, beans in the project
  - interface IAccount
  - class Account
    - accid, type, balance
    - constructors/getter/setters
    - deposit()/withdraw()
- create spring xml file with bean(3.0), context(3.0), aop(3.0).
- into spring xml file add usual tags <context:component-scan ... />

- @Component

```
@Aspect
class AccountAspects {
	@Before("execution (* pkg.Account.set*(..))")
	public void logBefore(JoinPoint pt) {
		System.out.println("BEFORE : " + pt.getSignature());
	}
	@After("execution (* pkg.Account.get*(..))")
	public void logBefore(JoinPoint pt) {
		System.out.println("AFTER : " + pt.getSignature());
	}
	@Around("execution (* pkg.Account.withdraw(..))")
	public Object monitorTransaction(ProceedingJoinPoint pt pt) throws Throwable {
		IAccount acc = (IAccount)pt.getTarget();

		System.out.println("Balance Before : " + acc.getBalance());

		Object res = pt.proceed();

		System.out.println("Balance After : " + acc.getBalance());

		return res;
	}
}
```

- in spring xml file:
  <aop:aspectj-autoproxy/>
- Spring implementation of AOP is based on open source AOP project "AspectJ".

+ **Spring Web MVC Architecture:**
- Used for implementing MVC into web applications.
+ MVC components are as follows:
+ View:
- implemented as JSP pages
- The JSPs can contain standard tags, JSTL tags, Velocity or any other third tag libraries.
- Also provide support for SPEL and JSP EL.
- Internally views are resolved by "ViewResolver" class.
+ **Controller:**
- The front controller is implemented by the spring mvc in form of servlet "DispatcherServlet" (need to make entry into web.xml).
- This controller is responsible for navigation control (flow).
- The controller is associated with a xml file in WEB-INF folder named as <spring-servlet-name>-servlet.xml.

web.xml

```
<servlet>
    <servlet-name>spring</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>spring</servlet-name>
    <url-pattern>/*</url-pattern>
</servlet-mapping>
```

- In this case, spring xml configuration should be created in WEB-INF with name spring-servlet.xml.
- For simplest application this file may have beans(3.0), context(3.0), mvc(3.0)

- The programmer can create one or more conroller classes.
- These controller classes are used to bind Model and View.
- Also these classes will contain one more request handlers (methods) which in turn execute the business logic typically from the service class (@Service).
- The request handlers will be resolved using "HandlerMapping" object.

+ **Model:**
- In spring Model is simple POJO classes.
- They are used to transfer data between View and Controller.

+ **Spring Web MVC Architecture (Flow)**
http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/images/mvc.png

+ **Steps for Spring Hello Web MVC:**

- create new "dynamic web project" e.g. spring_mvc
- in "WEB-INF/lib" folder add all spring jars (basic, mvc, tiles, hibernate, aop, etc).
- in web.xml make entry for spring front controller - DispatcherServlet
- in WEB-INF create spring config file (spring-servlet.xml) with beans(3.0), context(3.0) and mvc(3.0)
- in spring xml file:

```
<context:component-scan .../>
<mvc:annotation-driven/>
<bean id="viewResolver"
class="org.springframework.web.servlet.view.UrlBasedViewResolver">
        <property name="viewClass"
        value="org.springframework.web.servlet.view.JstlView"/>
        <property name="prefix" value="/" />
        <property name="suffix" value=".jsp"/>
</bean>
```

****

- in WebContent, create "first.jsp"

```
<h2>First Page</h2>
<a href="/spring_mvc/link_second">Second Page</a>
```

- in WebContent, create "second.jsp"

```
<h2>Second Page</h2>
```

- in src,

```
@Controller
class HelloController {
        @RequestMapping("/link_second")
        public String handleSecondLinkRequest() {
                return "second";        // name of next jsp
        }
}
```

****

- In WebContent/pages, create "third.jsp"

```
<h2>Third Page</h2> <hr/>
<a href="/spring_mvc/link_fourth">Next Page</a>
```

- In WebContent/pages, create "fourth.jsp"

```
<h2>Fourth Page</h2> <hr/>
Welcome Again!! <br/>
Today : ${today}
Book Details : ${bk.bookid}, ${bk.name}, ${bk.price}
```

- in HelloController class:

```
@RequestMapping("/link_fourth")
public String handleFourLinkRequest(Model model) {
        String curDate = new Date().toString();
```

```
                model.addAttribute("today", curDate);
                Book b = new Book(101, "book name", 123.50);
                model.addAttribute("bk", b);
                return "pages/fourth";
        }
```

****

1. in WebContent/caps -> index.jsp

```
<a href="/spring_mvc/input_link">Input Page</a>
```

2. CapModel class: (to add as model attribute)
   - private String inText;
   - constructor, getter, setter
3. CapsController class:

```
@RequestMapping("/input_link")
public String gotoInputPage(Model model) {
        CapModel cm = new CapModel();
        model.addAttribute("cm", cm);
        return "caps/input";
}
```

4. in WebContent/caps -> input.jsp

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

<form:form modelAttribute="cm" method="post" action="/spring_mvc/output_link">
        Enter Text : <form:input path="inText"/>
        <input type="submit" value="Submit"/>
</form:form>
```

5. CapsController class:

```
@RequestMapping("/output_link")
public String gotoOutputPage(Model model, @ModelAttribute("cm") CapModel cm) {
        String text = cm.getInText();
        text = text.toUpperCase();
        model.addAttribute("outputText", text);
        return "caps/output";
}
```

6. in WebContent/caps -> output.jsp

```
Input Text : ${cm.inText}
Caps Text : ${outputText}
```

+ Request handler method:

- The request handler methods (annotated with @RequestMapping) can have very flexible signature.
- It can take arguments of different types in any order:
  - Model / ModelMap / Map - data carried from controller to view or vice versa.
  - argument with @ModelAttribute - for easy access of model data.

- HttpSession - session object
- HttpServletRequest, HttpServletResponse
- argument with @CookieValue("cookieName") - for easy access of cookie data
- BindingResult - to add validation errors (usually written as last argument)

- It can also have different return types:
  - String - return view (jsp) name
  - ModelAndView - return ModelAndView object containing view name, model attr name, model attr object.
  - View - return View object containing jsp name

* For redirection from req handler method to another req handler method use "redirect:reqUrl" as view name.

```
@RequestMapping("/one")
String handleOne() {
	// ...
	return "nextjsp";
}
@RequestMapping("/two")
String handleTwo() {
	// ...
	return "redirect:one";
}
```

+ Spring 3 - Tiles 2 Steps:

1. In the spring project, ensure that tiles jars are present. tiles-api-2.2.2.jar, tiles-core-2.2.2.jar, tiles-jsp-2.2.2.jar, tiles-servlet-2.2.2.jar, tiles-template-2.2.2.jar, commons-digester-2.0.jar, commons-beanutils-1.8.0.jar

2. Configure TilesView url based resolver and tiles configurer in the spring config file.

```
<bean id="tilesConfigurer"
	class="org.springframework.web.servlet.view.tiles2.TilesConfigurer">
	<property name="definitions">
		<list>
			<value>/WEB-INF/tiles.xml</value>
		</list>
	</property>
</bean>

<bean id="viewResolver"
	class="org.springframework.web.servlet.view.UrlBasedViewResolver">
	<property name="viewClass"
		value="org.springframework.web.servlet.view.tiles2.TilesView" />
</bean>
```

3. Create a folder under WEB-INF and create master pages (templates) as well as header, footer and menu pages in it.

Note that master pages have <tiles:insertAttribute/> tags in which "name" attribute represent some name (linked with tiles.xml).

4. Create the content pages in the same folder or seperate folder. Note that content pages contains simple spring GUI stuff. No need to repeat <html> or <body> tag again as they are already included into master pages.

5. Create tiles.xml under WEB-INF folder. This file is linkage between master page and content pages.

```
<definition name="headerfooter.definition" template="/WEB-INF/jsp/headerfooter_master.jsp">
    <put-attribute name="title" value="" />
    <put-attribute name="header" value="/WEB-INF/jsp/header.jsp" />
    <put-attribute name="body" value="" />
    <put-attribute name="footer" value="/WEB-INF/jsp/footer.jsp" />
</definition>
```

The above tag defined master page in tiles.xml. Note that master page definition has got a "name" which will be used to define content pages. Also <put-attribute> "name" is mapped to <tiles:insertAttribute> "name" in the master pages.

```
<definition name="login" extends="headerfooter.definition">
    <put-attribute name="title" value="Login" />
    <put-attribute name="body" value="/WEB-INF/pages/login.jsp" />
</definition>
```

The above tag defined content page in tiles.xml. Note that content page "extends" from master page and <put-attribute> override the values given in master page definition. Finally "name" in the definition of content page will identify the "view" from the controller.

* String (ViewName) returned from controller will represent tile-defn in tiles.xml

+ Spring 3 - Hibernate 3 Steps:

1. In the spring project, ensure that hibernate jars are present.
2. configure hibernate into hibernate.cfg file - dialect, show_sql, entity classes
3. configure datasource in spring xml file

```
<bean id="dataSrc" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
    <property name="driverClassName" value="oracle.jdbc.OracleDriver" />
    <property name="url" value="jdbc:oracle:thin:@pdc1:1521:oracle" />
    <property name="username" value="smita" />
    <property name="password" value="smita" />
    <property name="initialSize" value="1" />
```

```
                <property name="maxActive" value="2" />
            </bean>
```

4. configure hibernate in spring xml file

```
            <!-- hibernate configuration -->
            <bean id="hbSF"
class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean">
                <property name="dataSource" ref="dataSrc"/>
                <property name="configLocation" value="classpath:hibernate.cfg.xml"/>
            </bean>

            <!-- hibernate transaction configuration -->
            <bean id="txManager"
class="org.springframework.orm.hibernate3.HibernateTransactionManager">
                <property name="sessionFactory" ref="hbSF" />
            </bean>

            <tx:annotation-driven transaction-manager="txManager"/>
```

5. in dao class, autowire hibernate SessionFactory "hbSF" and access current hibernate session to perform hibernate operations. Note that, the dao class or its methods should be marked as @Transactional.

+ Spring 3 Security Steps:

1. In the spring project, ensure that tiles jars are present.

spring-security-acl-3.1.0.RELEASE.jar, spring-security-config-3.1.0.RELEASE.jar, spring-security-core-3.1.0.RELEASE.jar, spring-security-crypto-3.1.0.RELEASE.jar, spring-security-taglibs-3.1.0.RELEASE.jar, spring-security-web-3.1.0.RELEASE.jar

2. In web.xml, configure spring application context

```
            <!-- Spring Application Context -->
            <listener>
                <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
            </listener>
            <context-param>
                <param-name>contextConfigLocation</param-name>
                <param-value>
                    /WEB-INF/spring-servlet.xml
                </param-value>
            </context-param>
```

3. Also, in web.xml configure spring security filter

```
            <filter>
                <filter-name>springSecurityFilterChain</filter-name>
```

```
            <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
        </filter>

        <filter-mapping>
            <filter-name>springSecurityFilterChain</filter-name>
            <url-pattern>/*</url-pattern>
        </filter-mapping>
```

4. Create LOGIN database :

CREATE TABLE spring_users (USER_ID INT(10) PRIMARY KEY, USERNAME VARCHAR(45), PASSWORD VARCHAR(45), ENABLED tinyint(1));

CREATE TABLE spring_roles (USER_ROLE_ID INT(10) PRIMARY KEY, USER_ID INT(10) REFERENCES spring_users(USER_ID), AUTHORITY VARCHAR(45));

5. In spring conf file, configure spring security

```
<security:http auto-config="true">
    <security:intercept-url pattern="/book/*" access="ROLE_USER" />
    <security:form-login login-page="/login" default-target-url="/book/newcart"
        authentication-failure-url="/login" />
    <security:logout logout-success-url="/logout" />
</security:http>

<security:authentication-manager>
  <security:authentication-provider>
    <security:jdbc-user-service data-source-ref="dataSrc"
      users-by-username-query="select username,password,enabled from spring_users
    where username=?"
      authorities-by-username-query="select u.username, ur.authority from spring_users u,
    spring_roles ur where u.user_id=ur.user_id and u.username=?"/>
  </security:authentication-provider>
</security:authentication-manager>
```

6. Create login.jsp with action "j_spring_security_check" and html textboxes as "j_username" and "j_password".

+ Java Server Faces:

- JavaServer Faces (JSF) is a Java specification(std part of J2EE specs , J2EE 1.5 onwards) for building component-based user interfaces for web applications.
- It's a standard Java EE web mvc framework.
- Well-designed and easy-to-use component-based web framework.

- JSF is based on well established MVC(Front Servlet Controller) design pattern.
- There are as many popular JSF implementations, namely Oracle Mojarra and Apache MyFaces & PrimeFaces, RichFaces etc.
- It helps to create web applications with server-side user interfaces (UIs).

+ JavaServer Faces technology consists of the following:

1. An API for representing components and managing their state; handling events, server-side,validation, and data conversion; defining page navigation; supporting internationalization etc.
2. Tag libraries for adding components to web pages and for connecting components to server-side objects.

+ A typical JavaServer Faces application includes the following parts:

1. Web pages in which components are laid out.
2. JSF custom tags to add components to the web page
3. Managed beans, which are lightweight container-managed objects (POJOs) based upon dependency injection, lifecycle callbacks and interceptors.
4. Web deployment descriptor (web.xml file)
5. Optional, configuration files -- faces-config.xml file, to define page navigation rules and configure beans and custom components.
6. Custom objects --- custom components, validators , converters, or listeners --programmer created.
7 A set of custom tags for representing custom objects on the page.

+ With minimal effort, you can complete the following tasks.

1. Create a web page.
2. Drop components onto a web page by adding component tags.(from JSF tag libraries)
3. Bind components on a page to server-side data.
4. Wire component-generated events to server-side application code.
5. Save and restore application state beyond the life of server requests.
6. Reuse and extend components through customization.

+ Developing a simple JavaServer Faces application typically requires the following steps:

1. Create JSF enabled web application.
   Dynamic web project -- Choose configuration -- JSF 2.1 , disable libraries (since jsf libs will be added later to <lib>), configure Faces Servlet , additional url patterns can be added -- *.jsf, *.xhtml.

2. Change compiler compliance level to 1.7 & change from project facet -- Java 1.7

3. Copy jsf jars under <WEB-INF>/lib.

4. Developing managed beans(equivalent to Controller in Spring , actions in Struts or Java beans in Standalone MVC) eg : HelloBean Annotations-

```
@ManagedBean(name="nameOfBean/id", eager="def:false")
default scope --- request scope.

@ManagedBean(name="hello")
public class HelloBean {
        private String message;
        // constructor, getter/setters
}
```

5. Create web pages using component tags

```
<html lang="en"
        xmlns="http://www.w3.org/1999/xhtml"
        xmlns:h="http://java.sun.com/jsf/html">
      <h:head>
            <title>Facelets Hello </title>
      </h:head>
      <h:body>
            #{hello.message}
      </h:body>
</html>
```

6. Mapping the javax.faces.webapp.FacesServlet instance -- Is already done at the time of creating JSF enabled web application.

+ Some more JSF annotations.

1. Scope related Annotations
   - @RequestScoped-- def scope --current request only.
   - @ViewScoped -- Bean lives as long as user is interacting with the same JSF view in the browser window or tab. It gets created upon a HTTP request and get destroyed once user postbacks to a different view.
   - @SessionScoped -- current session
   - @ApplicationScoped -- entire application
   - @NoneScoped -- no scope
   - Custom scope can also be created.

2. Dependency Injection annotation -- to inject some other bean .

Eg -- In managed bean -- UserBean

```
@ManagedBean(name="ub")
public class UserBean {
        @ManagedProperty(value="#{adr}")
```

```
        private Address address;
        // ...
}
-- in another managed Bean address
@ManagedBean(name="adr")
public class Address {
        // .....
}
```

* If eager=true & scope=application, only then bean is instantiated at the application start up time.

+ JSF life cycle.

- It involves --- the client makes a request for the web page, and the server responds with the page.
- The lifecycle consists of two main phases: execute and render.

During the execute phase:

1. The application view is built or restored.
2. The request parameter values are applied.
3. Conversions and validations are performed for component values.
4. Managed beans are updated with component values.
5. Application logic is invoked.

* For a first (initial) request, only the view is built. For subsequent (postback) requests, some or all of the other actions can take place.

During the render phase:

1. the requested view is rendered as a response to the client. Rendering is typically the process of generating output, such as HTML or XHTML, that can be read by the client, usually a browser.

+ JSF Internals:

Any JSF application goes through the following stages when it is deployed on the server.

1. When the application is built and deployed on the the application is in an uninitiated state.
2. When a client makes an initial request for the hello.xhtml web page, the hello Facelets application is compiled.
3. The compiled Facelets application is executed, and a new component tree is constructed for the hello application and is placed in a javax.faces.context.FacesContext.
4. The component tree is populated with the component and the managed bean property associated with it, represented by the EL expression hello.message
5. A new view is built, based on the component tree.
6. The view is rendered to the requesting client as a response.
7. The component tree is destroyed automatically.
8. On subsequent (postback) requests, the component tree is rebuilt, and the saved state is applied.

+ JSF Advantages:

1. JSF is based on well established Model-View-Controller (MVC) design pattern.
2. Applications developed using JSF frameworks are well designed and easy to maintain than any other applications developed in JSP and Servlets.
3. JSF provides standard, reusable components for creating user interfaces for web applications.
4. JSF provides many tag libraries for accessing and manipulating the components.
5. JSF is a specification and vendors can develop the implementations for JSF.

+ Enterprise Java Beans

- Enterprise applications contains number of presentation elements, a lot of business logic (for different purposes) and one or more databases.
- Most of the times there is need of executing business logic from different presentation elements (like web applications, desktop applications or mobile applications). EJBs are used to serve this purpose.

- What is EJB?

EJB is a java class containing business logic, which runs on an application server as per user request. EJB objects are created and managed by the EJB container.

- Why use EJBs?(Benefits)

1. EJBs simplify the development of large, distributed applications -scalable,reliable,secure
2. EJB container provides system-level services(Primary services JNDI, conn. pooling, Transactions, security:authorization, authentication, persistence: JPA) to the bean developer can concentrate on solving business problems.
3. For thick clnts : as the EJBs rather than the clients contain the application's business logic, the client developer can focus on the presentation(view) of the client. As a result, the clients are thinner, especially suitable for clients that run on small devices.(embedded)
4. EJBs are portable & reusable components. The application assembler can build new enterprise applications from existing beans.-- compliant on any Java EE server(application server)

- There are different types of EJBs:

1. session beans :
   - bean objects are always accessed via interfaces. Any type of session bean can support three types of interfaces:
   1. @Remote: this interface is used to access bean methods from outside the ejb container.
   2. @Local: this interface is used to access bean methods from within the ejb container (from some other ejb).
   3. @EndPoint: this interface is used only for stateless beans to implement web services.

+ stateless bean:

Once an instance is in the Method-Ready Pool, it is ready to service client requests. When a client invokes a business method on an EJB object, the method call is delegated to any available instance in the Method-Ready Pool. While the instance is executing the request, it is unavailable for use by other EJB

objects. Once the instance has finished, it is immediately available to any EJB object that needs it. Stateless session instances are dedicated to an EJB object only for the duration of a single method call.

Life Cycle:

1. when request is made of stateless bean, if it "does not exist", ejb container creates object of the bean.
2. then it calls @PostConstruct method, if written. We can write initialization code here.
3. Now bean object go to "ready" state. Any business logic method can be executed only if bean is in ready state.
4. At the end of life cyle, ejb container calls @PreDestroy method. We can write de-initialization code here.
5. Then bean object go to "does not exist" state. Now it is ready for garbage collection.

```
@Remote
public interface MathIntf {
        int add(int a, int b);
}

@Stateless(mappedName="ejb/math")
public class MathImpl implments MathIntf {
        public int add(int a, int b) {
                return a + b;
        }
}

// client side
InitialContext ctx = new InitialContext();
// initialize ctx, if accessing from outside the application server
MathIntf obj = (MathIntf)ctx.lookup("ejb/math");
int res = obj.add(10, 20);
```

+ stateful bean:

Each SFSB is dedicated to one client for the life of the bean instance; it acts on behalf of that client as its agent. Stateful session beans are not swapped among EJB objects, nor are they kept in an instance pool like their stateless session counterparts. Once a stateful session bean is instantiated and assigned to an EJB object, it is dedicated to that EJB object for its entire lifecycle.

Life cycle:

1. when a request come from new client, new object of the bean in created by the EJB container.
2. then EJB container calls @PostConstruct method, if written.
3. now bean object is in ready state. Client can call any business logic methods.
4. In low memory situations, ejb container may store the bean object in some temporary storage. In this stage, bean object cannot be used. So this stage is called as "Passive" state. Mostly ejb containers use LRU algorithm to decide which bean to be passivated.
5. EJB container calls @PrePassivate method before passivating bean object, if written.

6. If client requests for passivated bean, ejb container bring the bean object into the memory and make it "ready" for user requested operation.

7. before activation, ejb container calls @PostActivate method, if written.

8. When user want to release the bean object (when no longer needed), he calls @Remove method, if written.

9. After this ejb container calls @PreDestroy method, if written.

10. Now the bean object is eligible for garbage collection.

11. If bean is not used for very long time, ejb container can destroy the bean.

```
@Remote
public interface MathIntf {
	void setNum1(int a);
	void setNum2(int b);
	int add();
}

@Stateful(mappedName="ejb/math")
public class MathImpl implments MathIntf {
	private int a, b;
	public void setNum1(int a) {
		this.a = a;
	}
	public void setNum2(int b) {
		this.b = b;
	}
	public int add() {
		return a + b;
	}
}

// client side
InitialContext ctx = new InitialContext();
// initialize ctx, if accessing from outside the application server
MathIntf obj = (MathIntf)ctx.lookup("ejb/math");
obj.setNum1(10);
obj.setNum2(20);
int res = obj.add();
```

2. message driven beans

- Message Driven Beans are used when we need one way communication (from client to MDB).
- Here client simple send data to MOM and continue its own work.
- MOM transfers that data to the interested/corresponding MDB.

+ The messaging service can have one of the following model:

1. Queue : Point To Point Model
   - Durable Message
   - Msg will be sent if MDB is available, If not available, it will be pending in Queue
2. Topic : Publisher Subscriber Model
   - Non-Durable Message
   - Msg will be sent if MDB is available; If not available, msg will be lost.

- MDB is Enterprise Java Bean (Java Class) which is interested in receiving message from MOM.

```
@MessageDriven
// ...
public class SampleMDB implements MessageListener {
        public void onMessage(Message message) {
                System.out.println("Message Received!");
        }
}
```

+ Life cycle of MDB:

1. After creating the object of mdb, ejb container calls @PostConstruct method, if written.
2. When MDB is ready, it can receive messages from MOM i.e. its onMessage() will be called.
3. At the end of life cycle, ejb container will call @PreDestroy method, if written.
4. Then MDB object is eligible for garbage collection.

* Note, most of the times ejb containers create a pool of mdb objects and when message arrives, one of the object's onMessage() method is called.

3. entity beans:

+ Till ejb 2.1, there were three of ejbs:
  - session beans, message driven, entity beans (db)
+ In ejb 3.0 specification, there are only two types of ejbs:
  - session beans, message driven
+ Instead of entity beans a new specification of db connectivity is added known as JPA 1.0. JPA is an ORM specification.

+ Java Persistence API:

- JPA is specification given by SunMicrosystem for ORM.
- It is just like hibernate (as learnt before), with main difference being it is a specification; while hibernate is a product. Hibernate is one of the implementor of this specification.

- JPA entity life cycle:

1. **New:** When an entity object is initially created its state is New. In this state the object is not yet associated with an EntityManager and has no representation in the database.
2. **Managed:** An entity object becomes Managed when it is persisted to the database via an EntityManager's persist method, which must be invoked within an active transaction.
3. **Detached:** A managed entity object can also be retrieved from the database and marked for deletion, by using the EntityManager's remove method within an active transaction. The

entity object changes its state from Managed to Removed, and is physically deleted from the database during commit.

4. **Removed:** The last state, Detached, represents entity objects that have been disconnected from the EntityManager. For instance, all the managed objects of an EntityManager become detached when the EntityManager is closed.

- **JPQL is a query language much similar to HQL.**

॥ श्री गणेशाय नमः ॥

SUNBEAM

Institute of Information Technology

Placement Initiative

C Language

1. Explain flow of compilation and execution of C program?

main.c → Preprocessor → main.i → compiler → main.asm → Assembler → main.obj → .lib (Library files)

main.c - Source code main.i - Intermediate file

2. List name of five keywords given by ANSI?

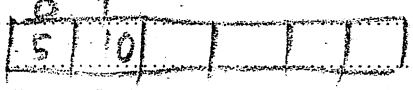
enum, signed, const, volatile, void

3. Write a program to find out maximum of three numbers using conditional operator?

```
int a,b,c;
(a>b)?(a>c?a:c):(b>c?b:c)
```

4. Why array index always starts with zero?

Normally, compiler converts arr notation into pointers e.g. arr[i] = *(arr[0]+1) = 10 i.e. location 1 in array, so we are skipping



element 5. So to access it,

5. How will you find out largest element without temporary variable in array?

```
for(i=0;i<3;i++)
{ for(j=1;j<3;j++)
{ if(a[i]<a[j])
{ a[i]=a[i]+a[j] //swap
a[j]=a[i]-a[j]
a[i]=a[i]-a[j]
printf("%d",a[0]);
```

6. What is the significance of storage class? List name of storage classes in C?

When we consider any variable we should consider its 2 parameters (characters) life and 2) scope. Life means existence of variable in memory and scope means its accessibility. 4 issues we need to consider about variables are 1) storage, 2) life, 3) Scope 4) default value

PRASANNA

macros can't be

C Language

7. What do you know about declaration and definition of variable?

Declaration of variable means we announce to compiler which variable we want to use in future and definition of variable means memory gets allocated to variable.

8. What is recursion and what is prerequisite for it?

In recursion we think of defining process in terms of itself and about terminating condition. These are prerequisite of recursion. recursion gives simplicity in code.

9. What is conditional compilation?

Conditional compilation is instruction given to compiler that compile specific code only if condition in which it is written is true otherwise don't. e.g. #if #else conditional compilation is normally used in device driver coding

10. What is the difference between function and macro?

fun are called at compile time and macros are processed at preprocessing level. fun are type safe and macro are not type safe. fun is slower and macro is fast. when we want to do big code use fun & for small code use macro. fun can be recursive & *

11. What is the significance of function calling convention? List name of function calling conventions? Which is default function calling convention?

Calling convention deals with argument pushing issues such as which fun will pop arguments (either calling fun or called fun). Argument popping responsibility is also called as stack cleanup. calling conventions are -

12. What is function activation record? What it contains?

function activation record is created on stack when fun is called. It contains local variables, formal arguments and return address (address of next instruction). when function is completed FAR is destroyed.

C Language

13. What is the difference between #include <xyz> and #include "xyz"

#include<xyz> this statement searches specified header file in standard library and #include "xyz" this will search specified header file in current project

14. What is the difference between typedef and #define?

typedef is keyword used to give another name to existing datatype typedef is handled by compiler e.g. size_t is used for unsigned int. #define is preprocessor directive which directs preprocessor to replace value of this

15. What is pointer? Which arithmetic operations we can perform on pointer?

pointer is one variable which holds address of memory location. Internally it's unsigned integer and never negative. pointer size is 4 bytes in visual studio. we can increment or decrement pointers. we can also subtract 2

16. Can you explain what is dangling pointer and memory leakage? How can we detect memory leakage?

dangling pointer points to memory which does not belong to current process. e.g. free(ptr) here ptr is dangling pointer. memory leakage means we allocated memory and forgot to deallocate it. this may decrease efficiency

17. What is function pointer? What is its use? Can you write code to declare n define function pointer with & without typedef ?

functionpointer returns address of first instruction of function to which it is pointing function pointer is used in IOT and IVT Syntax of functionpointer is void (*ptr)(int, int); and read it as pointer to the block of fun?

18. What do you know about heap corruption? Can u explain it using code?

Heap corruption is runtime error which occurs when we try to access memory which does not belong to memory allocated in dynamic memory allocating functions like malloc Stack corruption occurs when we try to access memory which is not allocated

19. Explain near, far and huge pointers

near ptr is 16 bit pointer to object contained in the current segment (2 byte)

20. What is difference between malloc and calloc?

Syntax of malloc() takes one argument and and calloc() takes 2 arguments (size of each element and how many elements) memory allocated by malloc() contains garbage and memory allocated by calloc contains 0

21. When we should use array and structure?

We should use array when we want to store data of same datatype. e.g. marks of students. And we should use structure when we can want to store logically related items of different datatypes together.

22. Why we cannot compare structure objects using == operator?

If our structure size is not in the multiple of word size then compiler adds stack bytes to make it in multiple of word size. == operator does bit by bit checking and checking stack bytes is meaningless

23. What is the difference between structure and union?

Structure size is addition of sizeof all data members in it and we can access any member at any time. In union size is size of that data member which is largest. We can access only one member of union at a time.

24. What is the difference between string length and string size?

25. What is variable argument function? Explain all the macros required to implement it with the help of example?

26. Write a swap function 1) Without temporary variable,2) Using macro 3) Using bitwise operator

27. Write a code to allocate and de-allocate memory for multidimensional array dynamically?

28. What do you mean by l-value and R-value?

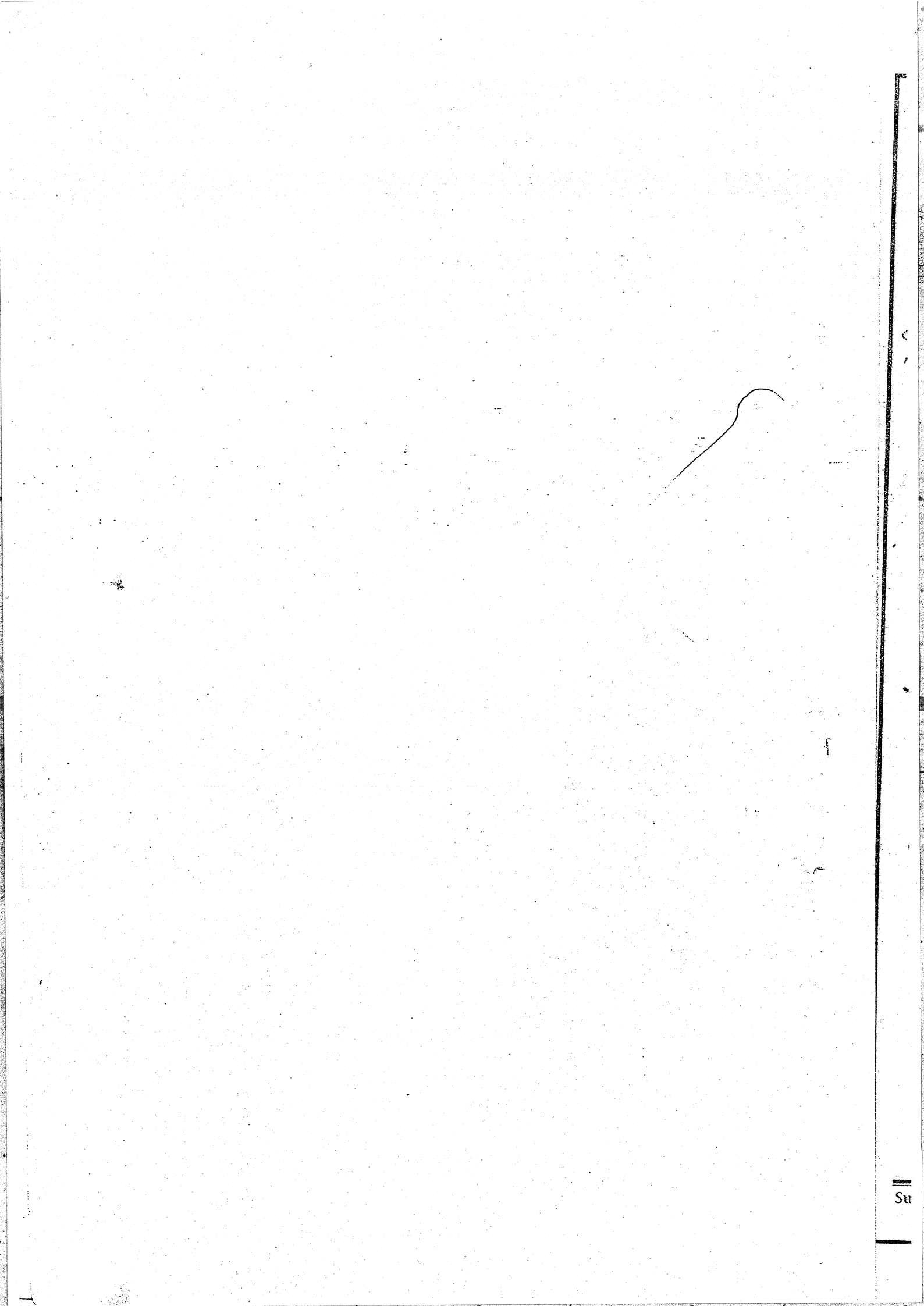
l means locator value or location value and r means reference value or result value. Rule says that LHS of assignment operator must be l value e.g. b+c = a is not allowed because b has location, char location but b+c has

29. What is the difference between getch, getche and getchar? Permemory location

getch() - Reads char from keyboard waits for enter key just get processed after getting any key pressed.
getche() - same as getch() but echoes char
getchar() - when you press any key these are kept in Buffer after hitting enter first char gets processed [illegible] screen

30. What is file? What is the difference between binary and text mode? List all binary and text mode? List all binary and text modes?

file is collection of data/info in secondary storage device. text mode deals with visual representation of data and binary file deals with memory representation of data. getc and fputc, fgets and fputs are text modes, fscanf and fprintf are also text mode. fread() and fwrite() are for binary mode



Data Structure

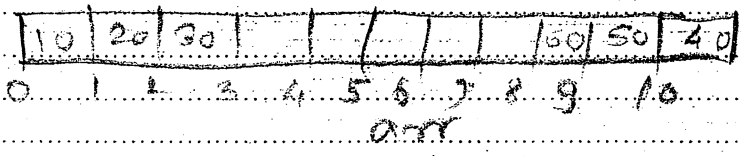
4. What is stack? Which are the applications of stack?

-Stack is a datastructure which can be implemented using array or linked list, operations

Applications of stack are evaluating infix, prefix or postfix expressions, conversions among them and process stack. When we call main internally it also goes on stack

2. How will you implement two stacks using single array?

We can implement 2 stacks using single array as



3. Convert a given infix expression into postfix and then evaluate it. (Write code).

4. What is Queue? Which are the types of queue?

Queue is datastructure (utility) which can be implemented using array or linked list. Queue can be operated on 2 ends as front and rear. There are 3 types of queues - 1) linear queue, 2) circular queue, 3) priority queue.

5. What is the queue full condition for circular queue?

There are 2 conditions which need to be checked for queue full condition and those are

1) q->front == -1 && q->rear == MAX-1

2) q->front == q->rear + 1 && q->front != -1

6. How can we implement stack and queue using linked list?

We can implement stack using linked list with operations ADD_first and Del_first and we can implement stack using linked list with operations ADD_last and del_last maintaining a pointers (head and tail in singly linked list).

7. Write binary search algorithm with and without recursion.

without recursion

```
while(
{   mid = left + right/2;
    if key == a[mid];
    return mid;
}
```

```
if(key < a[mid])
    right = mid-1;
else
    left = mid+1;
```

8. Which are the limitations of array over linked list?

We cannot dynamically grow or shrink array like linked list so once we allocate memory it is fixed. If we want to delete any inbetween element then we have to delete it and shift all others, but in linked list we can do this using 2 pointers only.

9. What is linked list?

Linked list is data structure with operations like ADD_last, ADD_FIRST, DEL_FIRST, DEL_LAST etc. We can traverse linked list only through serial access and we can dynamically grow or shrink memory required for linked list as per requirement

10. What is self-referential structure? What is its need?

Structure which contains pointer of its type is called self referential structure. Self referential structure is normally used in programming need where we want to refer next element which is of previous type only i.e. linked list Programming

11. Write a code to reverse linked list?

12. Write a code to reverse linked list using single pointer?

refer pages

13. How can we detect whether linked is linear or circular?

We can detect this in two ways - 1) by traversing whole linked list or by maintaining tail pointer. Complexity of traversal will be O(n) and complexity of maintaining tail pointer will be O(1) so second choice is better.

14. How will you write a code to create heterogeneous linked list?

Linked list in which each element/node is of different type is called heterogeneous linked list. ~~example~~ limitation of heterogeneous linked list is that we can only make it for union members.

15. How will you find out address of middle node in single traversal?

We can do this using 2 choices - 1) using count variable or by maintaining 2 pointers. In the first choice we can increment count as we traverse the whole linked list and calculate count/2 to get the mid element or

16. What is hash table? What is hash function? How it is designed?

17. What is collision and how to handle it?

18. What is bucket? What is load factor?

19. Explain tree traversal techniques?

There are 3 kinds of tree traversal techniques- 1) Preorder 2) Inorder 3) postorder we can implement above techniques using recursion or without using recursion, using recursion.

20. Which are the types of tree? Explain in brief?

Binary tree, Binary search tree, Complete Binary tree, Strictly binary tree, Full Binary tree, Full Binary tree, Almost complete binary tree, Threaded Binary tree, skewed binary tree, Balanced binary tree

21. What is AVL tree?

22. What is BFS & DFS?

BFS and DFS are searching techniques used to search node in tree. BFS means Breadth first Search and DFS means Depth first Search. BFS is also called as level wise Search

23. Write a code to implement merge sort?

24. Write a code to implement quicksort?

25. What is spanning tree?

26. What is binary search tree?

Binary search tree is a binary tree who in which left child node is always smaller than root node and right child node is always greater than or equal to root node. If we do inorder traversal of binary search

27. Write algorithm to implement TicTacToe game?

28. Write algorithm to implement sudoku game?

29. Write algorithm to implement maze game?

1

2.

3.

4.

5.

6.

Sur

OOP Concepts

1. What do you mean by major and minor pillar/system/element? Which are the major & minor pillars of oops?

There are 4 major and 3 minor pillars of OOPS. 4 major pillars are abstraction encapsulation, modularity and hierarchy. 3 minor pillars are 1) Typing (polymorphism) 2) concurrency 3) persistence

2. Explain abstraction and encapsulation with the help of real time example?

abstraction means getting essential details only. Encapsulation means hiding details which are unnecessary e.g. lift it has buttons to operate (i.e. abstraction) and internal details are hidden.

3. What is data security? How can we achieve it in C++?

Giving controlled access to data is called data security. We can declare it as private and give their access to member functions only.

4. What is modularity? What is the advantage of it? How can we achieve it in c++?

Process of dividing ~~code~~ compiler system in different files is called modularity. Advantage of modularity is complexity of code decreases. We can achieve this using namespace.

5. What is hierarchy? Which are the types of hierarchies? Explain all with the help of real time example?

Hierarchy means levels or order of abstraction main job of hierarchy is to achieve reusability, 1) has a - person has a heart 2) is a - HCl is a chemical 3) uses a - 4) creates a - different village creates current

6. What do you know about polymorphism? With the help of real time example, explain what is need of polymorphism?

polymorphism is ability of object of to take multiple forms. If we do not want to change interface and if we want to add new functionality then use polymorphism. It is used to reduce maintenance of system e.g. newspapers used for wrapping items also need also and

OOP Concepts

7. Which are the types of polymorphism and how can we achieve it in C++?

Runtime polymorphism and compile time polymorphism. If call to function gets resolved at runtime then it is Runtime polymorphism and if at compile time then it is compile time polymorphism we can achieve compile

8. Is it possible to achieve polymorphism in C? if yes then explain it with example?

Yes we can achieve Runtime Polymorphism in C using function pointer table and (i.e vtable in cpp) and address of specific function gets resolved at runtime

9. What is class and object explain with real time example?

Class is logical entity and object is physical entity. devices class and laptop is object class is blueprint / Template from which we can create object

10. Which are the characteristics of object?

characteristics of object are state, identity and behaviour responsibility state of object means value of its data members. member functions of class represent behaviour unique value used to distinguish between objects.

11. Why size of object of empty class is 1 byte?

class is logical object but object is physical entity. To distinguish it from class. Compiler vendors allocate 1 byte memory for object of empty class. according to standards size doesn't matter. So compiler vendors did optimization.

12. When we should use inheritance? Which are the types of inheritance?

If we want to add extra functionality to existing class or want to complete partially completed base class then we should use inheritance. Types are - 1) single inheritance 2) multiple inheritance, 3) multilevel 4) Hierarchical

OOP Concepts

13. What is function overloading? Which are the rules to overload function?

If implementation of function is logically equivalent or same then we should give same name to function but signatures should be different. Rules - 1) If no of parameter is same then no of parameters passed to the function must be different 2) If number of parameters passed to the function are same then type of at least

14. Why return type is not considered in function overloading?

catching values from function is not compulsory. We cannot consider return type in function overloading.

15. When to use abstract class, interface and non-abstract class (With Ref to Inheritance and polymorphism?)

16. What is the difference between interface and implementation inheritance?

interface inheritance means by without changing interface of system we can add extra functionality to system. this is called interface inheritance. e.g. product class. Implementation inheritance means if we want to reimplement function of base class into derived class with extra features then it is implementation inheritance

PRASANNA

* time polymorphism using - 1) Funct° overloading
2) operator overloading
3) Template

Runtime Polymorphism is achieved using Function overriding

C++

1. What is the difference between structure and class?

members of structure are by default public but members of class are by default private.

2. What is "this" pointer? Why static member function does not get this pointer?

this pointer is local pointer. this pointer is link between data member and member function. when we call member function using object then a this pointer is implicitly passed which holds address of current calling object. this is called this pointer.

3. What is destructor? When we should write destructor inside class?

destructor is used to deinitialize object. we can call destructor explicitly using object only in the case when we use placement new operator.

4. Can we overload destructor? Why?

No. we do not pass any parameter to destructor so we cannot.

5. What is the difference between malloc and new?

malloc() is function | new is operator
malloc() returns NULL | new throws exception bad- if it fails | alloc if it fails.
when we use malloc() | when we use new constructor
constructor do not get called | gets called

6. What is the difference between pointer and reference?

pointer can be initialized to NULL | we cannot initialize reference to NULL
we can create pointer to pointer | we cannot create reference to reference
we can create array of pointers | we cannot create array of references.

7. What is copy constructor? In which condition it gets call?

Copy constructor gets called when we try to assign value of state of ~~already~~ existing object to newly created object

8. What is shallow copy and deep copy?

Process of assigning value of object to another object is shallow copy. Deep copy needs to be created in 3 conditions - 1) If there is pointer data member in class 2) If there is user defined destructor in class and if copy of object need to be created

9. Which are the limitations of operator overloading?

We cannot overload size of, member selection (.), pointer to member selection (.*), conditional operators. Also the meaning of operator can change while operator overlooding. We cannot overload static_cast, typeid, dynamic_cast, const_cast & reinterpret_cast

10. In which condition we should overload index operator? Write a code to overload it?

When we want to treat object as array then we should overload index operator

11. What is the difference between copy constructor and assignment operator function?

copy constructor gets called when we assign value of already created object to newly created object and assignment operator function gets called when we try to assign already created object to already created object

12. What is smart pointer? Give any real time exmple of smart pointer?

Smart pointer is object treated as pointer. There are 3 smart pointer which is used to traverse ~~in~~ and lists collection. It is design pattern.

13. What is conversion function? Which are conversion functions in c++?

It is member function of class which is used to convert state of object of fundamental type into user defined type or vice versa following are fun conversion functions - 1) single parameter constructor, 2) Assignment operator fun, 3) Type conversion operator fun

14. Why static member function does not get this pointer?

Static member functions are designed to call with classname so they do not get this pointer

15. Which of the following functions cannot be declared as const? Why?

A. member functions B. global functions C. static functions D. friend functions

We cannot declare global functions and friend functions as const

Member functions in which we try to modify state of object those member functions we cannot declare as const

16. What is singleton class? Write a code for it? Give an example of its usage in program

Singleton class is such class of which we can create only one object Service request class.

17. Write a code to count number of instances created from a class?

We can use counter variable and one static function which will count total no of instances

18. What is diamond problem? How can we overcome it?

In case of diamond inheritance members of indirect base class gets inherited into indirect derived class multiple times that's why we cannot access these members of indirect base class inside member function of indirect derived class directly such problem created by hybrid inheritance is called Diamond Problem

19. What is object slicing? How can we avoid it?

If we try to copy state of object of derived class into state of object of base class then compiler copies values of only base class datamembers which is present inside derived class object into base class object such process is

20. What is virtual function? What is function overriding?

member function of derived class which is designed to call using pointer / Reference of base class is called virtual function of the class. Process of redefining / reimplementing virtual function of base class inside derived class with same signature is called function overriding.

21. Can we declare static member function as a virtual? Why?

Since we cannot declare static member function as virtual, we cannot override static member function in derived class since virtual functions are designed to call using base class pointer or base class reference variable only, and

22. Can we declare constructor as virtual? Why?

No, constructor is not designed to call using pointer or reference. So we cannot declare it as virtual. secondly if we declare constructor as virtual then its address will be stored in vtable and to store object address of vtable, constructor should be

23. What is need to declare destructor as virtual?

24. What is v-ptr and v-table?

V table will store starting address of all virtual functions of base class. vptr is datamember of derived class. It will store starting address of vptable So size of derived class object gets increased by 4 bytes

25. In C++, by default functions are not virtual, why?

As every function in c++ is not designed to call using pointer or reference. by default, every function is not made virtual in cpp. e.g. ~~member fun~~ non-member functions. It can be called using namespacename and scope resolution operator.

26. What is pure virtual function?

27. How to create final class in C++?

28. When we should declare function as a friend and class as friend?

If most of the ~~member~~ functions inside class are declared as friend of another class then declare it as friend. If less number of functions are declared as friend then do not do whole class as friend.

29. How can we restrict inheritance in C++?

30. What is runtime polymorphism? Explain with real-time example?

When call to function gets resolved at runtime then it is called Runtime Polymorphism.

31. Which are the casting operators given in C++? Explain use of each one with code example.

1) following are the casting operators in cpp. 1) static_cast 2) dynamic_cast 3) reinterpret_cast 4) const_cast. If we want to do type conversion between incompatible type then use ~~static_cast~~ reinterpret_cast

32. Why C++ do not support constructor chaining?

19) object slicing we can avoid object slicing by declaring base class as ~~abstract~~

21) Static member functions are not designed to call using base class pointer or base class reference variable we cannot declare static member function as virtual

22) called, so it will be called two times. and that is not valid

31) If we want to convert pointer to constant object to pointer to nonconstant object then use const_cast operator. dynamic_cast checks type conversion while doing conversion of incompatible types and overcomes limitation of static_cast So we use it. static_cast operator do not do type conversion and just checks inheritance that is its limitation

OS INTERVIEW QUESTIONS

1. Explain OS functionalities in detail.

Process management, memory management File management, CPU scheduling, IO management, Hardware Abstraction, Networking, Protection & security and User interfacing.

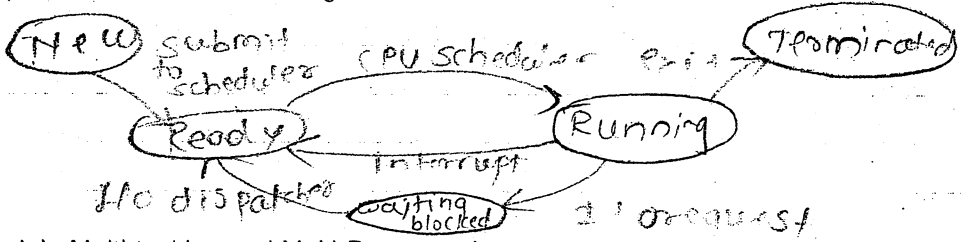
2. What is kernel?

The part of OS that performs basic minimal functionalities is called 'core OS' or kernel

3. Explain the output of "ls -l" command?

to see 'long' listing view, we can use ls -l command. for each file each shows read/write permissions, creation date and time, and its size and link count are also shown

4. Explain the Process state diagram in detail.



5. Explain Multi-tasking and Multi-Programming.

In Batch system, multiple programs can be loaded in memory. The no. of programs that can be loaded in memory is called degree of multiprogramming. Advantage of multiprogramming is increased CPU utilization. CPU time is shared among multiple processes in the main memory is called multitasking.

6. What is user space and kernel space? What is user mode and kernel mode?

memory area on RAM in which OS is present is called Kernel space and remaining all other area is called User space.

It is invoked
* implementation function from system call table and

OS INTERVIEW QUESTIONS

7. What is system call? How it is called?

System calls are the functions exposed by the kernel so that user programs can access kernel functionalities. System call contains APIs use software interrupt to switch to kernel mode, then SW into ISR select appropriate syscall *

8. Explain arguments and return value of read() and write() system calls.

int ret = read (fd, dest. addr, num of bytes) fd - file descriptor, dest. addr - here which data to be read and num. of bytes - how many to read,

int ret = write (fd, src. addr, number of bytes)

9. What is page fault ? What are page replacement algorithms ?

If CPU requests page that is not present in memory then page fault occurs.

FIFO, LRU, Optimal these are page replacement algorithms.

10. What is difference between fork() and exec() ?

fork() is system calls which creates child process by duplicating calling process. exec() is system call which loads new program in the calling process memory

11. What are different IPC mechanisms?

IPC means Interprocess Communication. It can be shared memory, message queue, socket, RPC etc.

12. What is pipe?

Pipe is special file which is used to communicate between two Processes. It is stream base unidirectional mechanisms. Pipe is internally implemented as kernel buffer in which data can be read / written

8) 1st and 3rd argument are same, both return 0 no. of bytes read. Anonymous on success and on failure return -1

OS INTERVIEW QUESTIONS

13. What is signal? Explain any five signals and their significance?

OS can send signal to another process or one Process can send signal to another process. No any data transfer occurs during sending signal. 1) SIGINT - If process do not handle it, it will be terminated. 2) SIGTERM 3) SIGKILL, 4) SIGSTOP. 5) SIG cont

14. What is interrupt? How it is handled in the system? How it is related to CPU scheduling?

When interrupt occurs then specific address of ISR is taken from IVT and it is executed before going out from ISR, scheduler is called. Scheduler checks if time of current process is completed, If yes then new process is scheduled by using dispatcher.

15. What is context switch?

If interrupt occurs when Process P1 was executing, its execution context was written back in its PCB. After that ISR will be invoked which will call Scheduler and scheduler will decide which process to execute next and will load its execution context in CPU registers.

16. What is vfork() ? and how it differs from fork () ?

vfork() was introduced in BSD unix. It creates PCB for child process and logical copy after that it pauses parent and executes child until exec() comes or exit comes. exec() checks size of a.out and creates actual copy in logical co

17. What is difference between symbolic link and hard link?

18. What is socket ? How it is created?

Socket is combination of IP address, and Port number. Socket is used for Remote Procedure call. Socket is communication end point. BSD UNIX developed it for first time. 2 types of socket

1) UNIX 2) Internet.

TCP UDP

**it has to create its physical copy

PRASANNA WADEKAR

OS INTERVIEW QUESTIONS

19. .What is deadlock? How you will avoid deadlock?

Deadlock occurs when following conditions hold true 1) No preemption 2) mutual exclusion 3) Hold & wait 4) circular wait. No preemption means Resource will not be released until task is completed. Mutual exclusion means Resources are *

20. What is Thread Synchronization? Differentiate semaphore and mutex ?

Semaphore is synchronization primitive where semaphore may be incremented by one process but another process may decrement it.

In mutex, process which has to access the mutex, same has to unlock it.

21. How will you compose command which will list only "permission attribute" and name of files from directory content?

22. What is difference between process and thread?

1) Process is container which has resource to execute threads.
2) there is parent child relationship between processes

1) thread is Executable entity which is present inside process and uses its resources.
2) There is no parent child relation ship between threads.

23.What is Copy on Write?

In modern sirux, we don't use vfork() but we use fork() itself. Here PCB of child process is created and page table of parent process is copied in that but actual copy is not created. until child performs write operation all goes well. moment it tries to write it is not shareable. Hold & wait means one process has hold resource and it is waiting for other. Circular wait means P1 has hold R1 and it is waiting for R2. R2 is hold by P2 and it is hold by P1. we can avoid deadlock using 2 methods 1) advanced announcement of resources required. 2) algorithm - safe state, Resource allocation graph &