Standard folder structure of webapps to deploy in tomcat server
==========

```
webapps(deployment folder)
      |
SecondApp
     |=> WEB-INF
                 |=> web.xml(deployment descriptor)
                 |=> classes
                              |=> *.class
     |=>  src/main/java
                 |=> .java
```

Any Servlet will be executed by the container with its life cycle actions

a. Loading
b. Instantiation
c. Initialization
d. RequestProcessing phase
e. De-Instantiation

What is the meaning of webapps(deployment folder)?
Once we start the server,server would automatically go to webapp's folder and scans all the project present inside
that folder and deploys the project in the execution area.(meaning ready for providing the service)
Since it scans for all the projects, we say webapp's folder as "Deployment folder".
Once the Tomcat engine loads all the project into execution area,It will create sepearte object for every project
called "ServletContext" object.
Tomcat engine also scans web.xml file given by the user w.r.t every project.
It read the url-pattern for all the dynamic resource for future usage.

How to send the request to any application?
using the url pattern
eg:
http://localhost:9999/[NameOfTheApplication/ContextRoot]/[url_pattern of the resource]

Assume the request is sent for a particular url_pattern, then what actions will be taken care by tomcat engine?
eg: http://localhost:9999/SecondApp/demo

Step1: Browser will send the request with the follwing url pattern
http://localhost:9999/SecondApp/demo

Step2: HttpProtocol will create a HttpRequest object depending upon the Request_TYPE to carry the request
data from client to server.

Step3: Once the Protocol hands over the HttpRequest Object to the tomcat server, server will pick up the
RequestLine from the HttpRequestObject and take only ContextName/urlpattern for furthe processing.

Step4: Depending upon the dynamic resource identified by urlpattern,tomcat engine will hand over the control
to container for execution.
Container will scan for web.xml file and identifies for a particular url_pattern which servlet should be

executed,based on that the servlet will be executed by the container.

Step5: As per the container life cycle actions, the requested urlpattern servlet will be executed.

1. Servlet Loading(.class file loading)
   Class c = Class.forName("SecondServlet");
2. Servlet Instantiation(for loaded class create an Object)
   SecondServlet obj=(SecondServlet)c.newInstance();
3. Servlet Initalization(for the created object inject the required values)
   obj.init(ServletConfig config);
4. Request Processing phase(for client request this method will be called)
   obj.service(ServletRequest request,ServletResponse response);

Note:
To get the inputs supplied by the user to the Servlet, we need to use "ServletRequest" Object.
To write the output from the application to the browser, we need to use "ServletResponse" Object.
Inside the ServletResponse object we have empty "PrintWriter" object,using which we need to write the
Output to browser window.
If we use System.out.println() in the request processing logic, then output will be available on the console of
the tomcat engine.

5. Once the server is stopped(undeployment)/for the same resource if the request wont' come for some
   period of time automatically container will execute "De-Instantiation" event.
   obj.destroy()

Before sending the request, we need to keep the compiled code in WEB-INF/classes folder

```
C:\Tomcat 9.0\webapps\SecondApp>set path=C:\Program Files\Java\jdk1.8.0_202\bin
C:\Tomcat 9.0\webapps\SecondApp>set classpath=C:\Tomcat 9.0\lib\servlet-api.jar
C:\Tomcat 9.0\webapps\SecondApp>javac SecondServlet.java
```

Once the compiled code available then , we need to start the server

a. c:\tomcat9.0\bin\tomcat9.exe

Output For First Request

```
==================
SecondServlet .class file is loading...
SecondServlet Object is instantiated...
Servlet initialziation...
Servlet Request Processing ...
```

Output for Second Request

```
======================
Servlet Request Processing ...
```

As we noticed above, the processing time for second request is less when compared to first request becoz only
requestprocessing logic is executed.

How to maintain the uniformity of response time for all the request?
Ans.To uniform response time for all the request we need to configure the tomcat engine.
Tomcat engine can be configured in 2 ways
a. XML(<load-on-startup>any+ve number<load-on-startup>)
b. Annotation

To maintain the uniform response time we need to use <load-on-startup> tag.

```
<web-app>
      <servlet>
            <servlet-name>DemoServlet</servlet-name>
            <servlet-class>SecondServlet</servlet-class>
            <load-on-startup>10</load-on-startup>
      </servlet>
      <servlet-mapping>
            <servlet-name>DemoServlet</servlet-name>
            <url-pattern>/demo</url-pattern>
      </servlet-mapping>
</web-app>
```

At the time of Server startup
==========
SecondServlet .class file is loading...
SecondServlet Object is instantiated...
Servlet initialziation...

Output for Firstrequest
==========
Servlet Request Processing ...

Output for Second Request
==========
Servlet Request Processing ...

Servlet Code w.r.t Annoatation
==========
Use the following annotations on the top of Servlet class as shown below.

```
@WebServlet(urlPatterns="/test",loadOnStartup = 10)
```

Limitation of implementing Servlet interface
==========
If we create a servlet using Servlet(I), then it is mandatory for us to give the implementation for all the methods
of the interface whether it is required or not.
Becoz of this length of the code would increase and it decreases the readability.
To Overcome this problem we need to use "GenericServlet".

GenericServlet has already implemented Servlet Interface and it gives body for all the methods of Servlet interface
except service().
if we use GenericServlet to create a Servlet, then we need to give body only for service() as a result of which the
lines of code would be less, which increase the readablity of the application.

Compiled from "GenericServlet.java"

```
public abstract class GenericServlet implements Servlet,ServletConfig,Serializable
{
  public GenericServlet();
  public void init(ServletConfig config) throws ServletException;
  public void destroy();
  public ServletConfig getServletConfig();
  public ServletContext getServletContext();
  public String getServletInfo();

  public void init() throws javax.servlet.ServletException;
  public abstract void service(ServletRequest req, ServletResponse resp) throws
ServletException,IOException;
  public java.lang.String getServletName();
}
```

Note:
GenericSevlet is an best example for "Adapter class design pattern".
init() is overloaded in GenericServlet.

Note:
By default response type/content type is "text/html".

Code to demonstrate the creation of Servlet using GenericServlet
==========

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.annotation.*;

@WebServlet(urlPatterns="/disp")
public class FourthServlet  extends GenericServlet
{
      public  void service(ServletRequest request, ServletResponse response) throws

            ServletException,IOException
      {
                  PrintWriter out = response.getWriter();
                  out.println("<h1 style='color:blue;'>Writing Servlet using
Generic Servlet is easy....</h1>");
                  out.close();
      }
}
```

Behind the scenes
==========

In the above code 2 .class files will be used
a. FourthServlet.class
b. GenericServlet.class

=> Loading :: Container will load FourthServlet.class file for the url pattern ("/disp")
=> Instantiation :: Container will create an Object for FourthServlet.class
=> Initialization :: Container will call init(),First it will check in FourthServlet.class if not, it would check in GenericServlet.
init() is available inside GenericServlet but it has 2 methods with the name
init(SC config)
init()
container will call init(SC config) which internally makes a call to init().

Can we override the init logic?
We can override, but it is a good practise to override only init(),but not init(SC config) becoz config is local variable
in init(SC config),and the config variable memory would be gone once the control comes out of the init(SC config)
so better override init() but not init(SC config).

Code in GenericServlet
==========
```
public abstract class GenericServlet implements Servlet,ServletConfig,Serializable
{
        private transient SC config;
        public void init(SC config)throws SE
        {
                this.config=config;
                init();
        }
        public void init() throws SE
        {
                ;;;;;;
        }
        ;;;;;;;;
}
```

=> RequestProcessing phase :: Container will call service(req,resp) to provide response to the client.
First it will check in FourthServlet.class if not, it would check in GenericServlet.
service(req,resp) is available inside GenericServlet as abstract and we need to give the body
of this method inside FourthServlet as shown in the above program.

=> ServletDeInstantion=> Contianer will call destroy() to perform De-Instantion action.
First it will check in FourthServlet.class if not, it would check in GenericServlet.
destroy() is not avaialble in FourtServlet.class,so it would take from GenericServlet.class and it will execute.

Note:
1. If our servlet class does not contains init() method
    a. GS: init(SC)
    b. GS: init()
    c. US: service(req,resp)

2. If our servlet class contains init(SC) method
    a. US: init(SC)
    b. US: service(req,resp)

3. If our servlet class contains init() method
    a. GS: init(SC)
    b. US: init(SC)
    c. US: service(req,resp)

why 2 init(),init(SC config) in GenericServlet?

```
            init(SC config) -> container
            init() -> developer

Which init method is best suited for developer?
            init() => best suited for writing initialziation logic.
```