**JDBC**

*The JDBC API is a Java API that can access any kind of tabular data, especially data stored in a relational database.*

*JDBC helps you to write Java applications that manage these three programming activities:*

1. *Connect to a data source, like a database*
2. *Send queries and update statements to the database*
3. *Retrieve and process the results received from the database in answer to your query*



*// JDBC Product Components*

*// JDBC includes four components:*

*// The JDBC API —  The JDBC™ API provides programmatic access to relational data from the Java™ programming language. Using the JDBC API, applications can execute SQL statements, retrieve results, and propagate changes back to an underlying data source. The JDBC API can also interact with multiple data sources in a distributed, heterogeneous environment.*

*// The JDBC API is part of the Java platform, which includes the Java™ Standard Edition (Java™ SE ) and the Java™ Enterprise Edition (Java™ EE). The JDBC 4.0 API is divided into two packages: java.sql and javax.sql. Both packages are included in the Java SE and Java EE platforms.*

*// JDBC Driver Manager —  The JDBC DriverManager class defines objects which can connect Java applications to a JDBC driver. DriverManager has traditionally been the backbone of the JDBC architecture. It is quite small and simple.*

*// The Standard Extension packages javax.naming and javax.sql let you use a DataSource object registered with a Java Naming and Directory Interface™ (JNDI) naming service to establish a connection with a data source. You can use either connecting mechanism, but using a DataSource object is recommended whenever possible.*

*// JDBC Test Suite —  The JDBC driver test suite helps you to determine that JDBC drivers will run your program. These tests are not comprehensive or exhaustive, but they do exercise many of the important features in the JDBC API.*

*// JDBC-ODBC Bridge —  The Java Software bridge provides JDBC access via ODBC drivers. Note that you need to load ODBC binary code onto each client machine that uses this driver. As a result, the ODBC driver is most appropriate on a corporate network where client installations are not a major problem, or for application server code written in Java in a three-tier architecture.*

*// This Trail uses the first two of these four JDBC components to connect to a database and then build a java program that uses SQL commands to communicate with a test relational database. The last two components are used in specialized environments to test web applications, or to communicate with ODBC-aware DBMSs.*

*Driver software: which convert java calls to database calls and vice versa*

*Connectivity: connection between java application and database*

*Statement object: to send some requirement (SQL Query) and get response from database in a result set*

*JDBC:--- is a technology to communicate with database java applicaton*

*JDBC Interview Questions:-*

*-----------------*

*1. Explain what do you mean by JDBC?*

*2. What are the steps to connect to a database in java?*

*3. What are the JDBC API components?*

*4. How to you load the drivers in JDBC?*

*5. What is the purpose of JDBC ResultSet interface?*

*6.  Explain how you can establish a connection ?*

*7. What are the different types of statements in JDBC ?*

*8.  Have you used prepared statements? Where have you used prepared statements ?*

*9.  How do you create JDBC statements ?*

*10. What is the difference between a Statement and a PreparedStatement?*

*11. What are callable statements ?*

*12. What are types of JDBC drivers?*

*13. How do you retrieve data from a result set, explain with an examplet?*

*14.  Explain about stored procedure ?*

*15. When do we look for batch updates ?*

*16. What is the return type of Class.forName() method?*

*17. How can we execute stored procedures using CallableStatement?*

*18. What is the role of the JDBC DriverManager class?*

*19. What are the functions of the JDBC Connection interface?*

*20. What is the major difference between java.util.Date and java.sql.Date data type?*

**1. What do you mean by JDBC?**

**JDBC (Java Database Connectivity)** is an API in Java that allows Java programs to connect to and interact with databases.

✅ **Example**: You can use JDBC to run SQL queries from a Java program.

**2. Steps to connect to a database in Java?**

1. Load the driver
2. Create a connection
3. Create a statement
4. Execute the query
5. Process the result
6. Close the connection

✅ **Example**:

java

CopyEdit

Class.forName("com.mysql.cj.jdbc.Driver");

Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb", "user", "password");

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery("SELECT \* FROM students");

while(rs.next()) {

System.out.println(rs.getString("name"));

}

conn.close();

**3. JDBC API Components**

* **DriverManager**
* **Connection**
* **Statement**
* **PreparedStatement**
* **CallableStatement**
* **ResultSet**

**4. How do you load the drivers in JDBC?**

Using Class.forName() method.

✅ **Example**:

java

CopyEdit

Class.forName("com.mysql.cj.jdbc.Driver");

**5. Purpose of JDBC ResultSet interface?**

To hold and process the result returned by SQL queries (SELECT).

✅ **Example**:

java

CopyEdit

ResultSet rs = stmt.executeQuery("SELECT \* FROM employees");

while (rs.next()) {

System.out.println(rs.getInt("id") + " " + rs.getString("name"));

}

**6. How can you establish a connection?**

By using DriverManager.getConnection().

✅ **Example**:

java

CopyEdit

Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb", "root", "password");

**7. Different types of statements in JDBC**

* Statement
* PreparedStatement
* CallableStatement

**8. Have you used PreparedStatement? Where?**

Yes, used it to avoid SQL injection and for dynamic SQL.

✅ **Example**:

java

CopyEdit

PreparedStatement ps = conn.prepareStatement("SELECT \* FROM users WHERE email = ?");

ps.setString(1, "abc@example.com");

ResultSet rs = ps.executeQuery();

**9. How do you create JDBC statements?**

Using Connection object.

✅ **Example**:

java

CopyEdit

Statement stmt = conn.createStatement();

**10. Difference between Statement and PreparedStatement**

| **Statement** | **PreparedStatement** |
| --- | --- |
| Simple SQL | Precompiled SQL |
| Slower for repeated use | Faster for repeated use |
| Prone to SQL injection | Safe from SQL injection |

**11. What are CallableStatements?**

Used to execute **stored procedures** from Java.

✅ **Example**:

java

CopyEdit

CallableStatement cs = conn.prepareCall("{call getEmployee(?, ?)}");

cs.setInt(1, 101);

cs.registerOutParameter(2, Types.VARCHAR);

cs.execute();

**12. Types of JDBC drivers**

1. **JDBC-ODBC bridge driver**
2. **Native-API driver**
3. **Network Protocol driver**
4. **Thin driver (Pure Java)** ✅ Most common

**13. How to retrieve data from ResultSet?**

✅ **Example**:

java

CopyEdit

ResultSet rs = stmt.executeQuery("SELECT \* FROM users");

while(rs.next()) {

System.out.println(rs.getInt("id") + ", " + rs.getString("name"));

}

**14. What is a stored procedure?**

A stored procedure is a group of SQL statements stored in the database that can be called by name.

✅ **Example** (MySQL):

sql

CopyEdit

CREATE PROCEDURE getUsers()

BEGIN

SELECT \* FROM users;

END

**15. When do we use batch updates?**

When we need to execute **multiple SQL statements together**, like inserting many rows.

✅ **Example**:

java

CopyEdit

Statement stmt = conn.createStatement();

stmt.addBatch("INSERT INTO students VALUES (1, 'John')");

stmt.addBatch("INSERT INTO students VALUES (2, 'Jane')");

stmt.executeBatch();

**16. Return type of Class.forName()**

It returns a **Class<?>** object.  
It’s used to load the class into memory (side-effect: driver gets registered).

**17. How to execute stored procedures using CallableStatement?**

✅ **Example**:

java

CopyEdit

CallableStatement cs = conn.prepareCall("{call getStudentName(?)}");

cs.setInt(1, 101);

ResultSet rs = cs.executeQuery();

**18. Role of DriverManager class**

Manages the list of database drivers and **establishes connections**.

✅ **Example**:

java

CopyEdit

Connection conn = DriverManager.getConnection(...);

**19. Functions of JDBC Connection interface**

* Create Statement
* Manage transactions
* Get metadata
* Close the connection

✅ **Example**:

java

CopyEdit

conn.setAutoCommit(false);

conn.commit();

conn.rollback();

**20. Difference: java.util.Date vs java.sql.Date**

* java.util.Date: General-purpose date/time
* java.sql.Date: Only contains date (no time), used for DB

✅ **Example**:

java

CopyEdit

java.sql.Date sqlDate = new java.sql.Date(new java.util.Date().getTime());



***SQL injection***

*How to Protect a JDBC Application Against SQL Injection?*

*What is SQL Injection ?*

*- One of the top 10 security attacks as per OWASP (Open Web Application Security Project)*

*- It is a type of cybersecurity vulnerability that allows an attacker to interfere with the queries an application sends to its database.*

*How It Works*

*If a web application improperly handles user input, an attacker can inject malicious SQL code into a query. This can allow them to:*

*View confidential data*

*Modify or delete data*

*Bypass login authentication*

*etc.*

*How it can happen in real world ?*

*One of the ideal situations is when an application asks the user for input such as username or user id. The application opened up a vulnerable spot there. The SQL statement can be run unknowingly. An attacker takes advantage by injecting a payload that to be used as a part of the SQL query and processed by the database.*

*eg in JDBC -*

*String sql= select \* FROM users where*

*user\_name = '" + uname + "' and password= '" + pass + "'";*

*Statement st=cn.createStatement();*

*stmt.executeQuery(sql);*

*This code is vulnerable to SQL injection attack because the input given to the SQL statement through the variable 'uname' and 'pass' can be manipulated*

*String sql= select \* from users where*

*user\_name = '" + uname + "' and password= '" + pass + "' OR 1;*

**

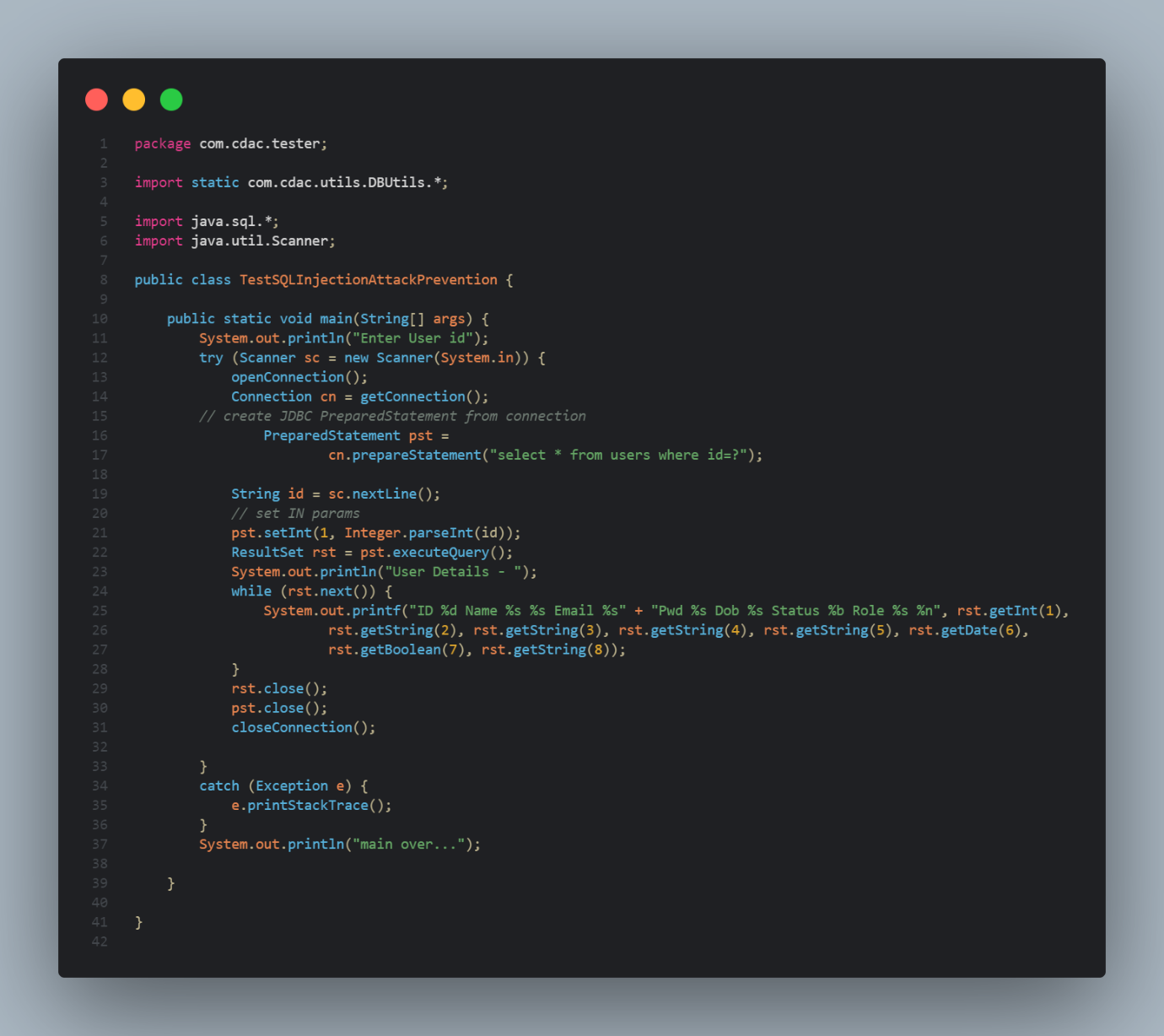
*This results in modifying the original SQL statement to a degree that enables one to bypasses authentication. This is a serious vulnerability and must be prevented from within the code.*

*Defence Against an SQL Injection Attack*

*Use PreparedStatement instead of a Statement. It sends a pre-compiled SQL statement to the database, rather than a string. This means that query and data are separately sent to the database.*

*This prevents the root cause of the SQL injection attack, because in SQL injection, the idea is to mix code and data wherein the data is actually a part of the code faked as data.*

*In PreparedStatement, there are multiple setType() methods, such as setString(). These methods are used to filter special characters such as a quotation contained within the SQL statements.*

**

DA2

*Today's topics*

*Revision*

*SQL Injection attack n it's prevention*

*CallableStatement in JDBC n execution of stored procedure*

*ResultSet Cursors*

*Transaction management overview in JDBC*

*Enter Java EE 8*

*HTTP request response flow*

*Dynamic web project structure*

*Java Servlets - concept , API n deployment*

*1. Revision of JDBC*

*2. For SQL injection ,*

*Refer -*

*WebJava\day2-data\day2\_help\SQL injection*

*readmes n code*

*3. Invocation of Stored Procedure / function*

*Development steps*

*3.1 Create CallableStatment*

*Use Connection i/f method*

*public CallableStatement prepareCall(String sql) throws SQLException*

*JDBC syntax for calling a procedure -{call procName(?,?,?,....?)}*

*sql syntax for calling a function -{?=call funcName(?,?,?,....?)}*

*{} => Esc sequence meant for JDBC driver to translate proc/func invocation in DB specific manner.*

*? => IN | OUT | IN OUT*

*3.2 For OUT n IN OUT parameters ,*

*Register them with JVM (i.e you will have to specify generic SQL type - available from constants in java.sql.Types class , before the execution)*

*API of CallableStatement*

*public void registerOutParameter(int parameterIndex,int sqlType) throws SQLException*

*3.3 Set IN parameters*

*Methods inherited from PreparedStatement*

*public void setType(int paramIndex,Type value);*

*3.4 Execute the procedure / function*

*public boolean execute() throws SQLException*

*Ignore return value here.*

*3.5 Read results from OUT parameters*

*Method of CallableStatement*

*public Type getType(int paramIndex) throws SQLException*

*paramIndex - index of OUT parameter.*

*type - JDBC data type (generic SQL type)*

*4. Default type of the ResultSet cursor*

*TYPE\_FORWARD\_ONLY => can traverse in forward direction only.*

*READ\_ONLY => can only use getters to read data from Result set*

*Using additional APIs , it can be changed to*

*SCROLLABLE\_TYPE n can be updated.*

*For more details – “day2\_help\regarding result sets”*

*5. Transaction(Tx) management overview*

*Steps*

*1. To begin a Tx , disable auto commit*

*Connection i/f method*

*public void setAutoCommit(boolean flag) throws SE*

*2. Wrap entire Tx in a try block.*

*2.1 At the end of try block , commit the Tx.*

*Connection i/f method*

*public void commit() throws SE*

*2.2 In a catch block , rollback Tx*

*Connection i/f method*

*public void rollback() throws SE*

*3. To continue after Tx , enable auto commit flag.*

*eg : connection.setAutoCommit(true);*

***Enter Web programming in Java***

*1. What is Java EE ?(Java Enterprise Edition)*

*Version - Java EE 8 (vendor - Oracle / Sun Microsystem)*

*Will be upgraded to Java EE 9 - Jakarta EE 9/10 (Maintained by Eclipse Foundation)*

*It is a set of specifications that extends the Java SE (Standard Edition) for additional enterprise features*

*eg - distributed computing and web services.*

*It's designed to provide a platform for building and deploying large-scale, multi-tiered, scalable, and secure network applications.*

*Which specifications ? (Rules or contract )*

*Specifications of primary essential services required for any enterprise application.*

*What is enterprise application ?*

*An enterprise application (EA) is a large software system platform designed to operate in a corporate environment . Typically , it is server side ,remotely deployed , transactional , multi threaded ,complex , scalable application.*

*It includes online shopping and payment processing, interactive product catalogs, computerized billing systems, security, content management, IT service management, business intelligence, human resource management, manufacturing, process automation, enterprise resource planning ....*

*The Java EE specifications include ---*

*Servlet API,JSP(Java server page) API,Security,Connection pooling ,EJB (Enterprise Java Bean), JNDI(Naming service -- Java naming & directory i/f),JPA(java persistence API),JMS(java messaging service),Java Mail, Java Server Faces , Java Transaction API, Webservices support(SOAP/REST) etc...*

*Vendor of Java EE specs - Oracle / Sun / Eclipse*

*Implementation - left to vendors (Java EE server vendors)*

*Java EE compliant web server - Apache supplied*

*- Tomcat (web server) 10.1.26*

*Services implemented --- servlet API,JSP API,Security,Connection pooling,JNDI(naming service)*

*Java EE complaint application server --- web container + EJB (enterprise java bean) container*

*+ ALL Java EE services implementation*

*Java EE server Vendors & Products*

*Apache -- Tomcat(web server) / Tomee (app server)*

*Oracle / Sun --- reference implementation --- Glassfish*

*Red Hat -- JBoss (wild fly)*

*Oracle / BEA -- Oracle weblogic*

*IBM -- Websphere*

***2. WHY Java EE***

*1. It can support different types of clients including*

*thin client(web client)*

*thick client --- java application client*

*smart clients -- mobile clients*

*2. Java EE based applications are* ***server independent****.*

*You can Create & deploy server side appln , on ANY* ***Java EE compliant*** *server .*

*It is guaranteed to produce the SAME results w/o any modifications on ANY other Java EE server*

*It’s the* ***separation*** *between specifications (supplied by Jakarta EE) n implementation is left to Java EE server vendors.*

*3.Java EE servers provide -*

*Ready made implementation of primary services, so that Java EE developer doesn't have to worry about it's implementation ,but rather can concentrate on actual business logic.*

*3. Layers involved in HTTP request-response flow (refer to day1-data\day1\_help\diags\request-response-flow.png)*

*4. What is dyn web application?*

*It is a server side appln , deployed on web server , meant for servicing typically web clients, using application layer protocol HTTP /HTTPS*

*(ref : diag request-resp flow)*

*URL :* [*http://www.mybank.com:8080/banking*](http://www.mybank.com:8080/banking)

*http : app layer protocol(scheme)*

[*www.mybank.com*](file:///E:\WPJ\Day%202\day2-data\day2-data\day2_help\www.mybank.com) *: DNS qualified host name /IP address (to resolve the host)*

*8080 : TCP port no (to identify the port --> service : web server)*

*/banking : path or URI (uniform resource identifier)*

*In Java , when you are creating a web app*

*URI : starts with context path (web context =web app) , typically set as web project name.*

*5. Objective*

*Creating & deploying a dyn web appln on Tomcat*

*Test HTML content*

*6. IDE automatically creates Java EE compliant web application folder structure .*

*Its details -- Refer to diag (Java EE compliant web app folder structure)*

*7. What is Web container (WC) & its jobs -*

*It is a Server side JVM residing within web server.*

*Its run-time environment for dynamic web components(Servlet & JSP,Filter) .*

*Jobs -*

*1. Creating Http Request & Http response objects*

*2. Controlling life-cycle of dynamic web components (manages life cycle of servlet,JSP,Filters)*

*3. Giving ready-made support for services - Naming,security,Conn pooling .*

*4. Handling concurrent request from multiple clients .*

*5. Managing session tracking*

*6. Manages state(session tracking)*

*7. Helps in page navigation*

*n many more...*

*8. What is web.xml ?--- Deployment descriptor one per web appln*

*created by - developer (with help of IDE)*

*where (location) – WEB-INF (private)*

*who reads it - WC*

*when --- deployment time (run server side app)*

*what does it consist of ?*

*Deployment instructions*

*eg - welcome page, servlet deployment tags, session configuration, security configuration etc.*

*9. Why servlets?*

*To add dynamic nature to the web application , performing these jobs*

*Job list*

*1. Request processing*

*2. B.L (Business Logic)*

*3. Dynamic response generation*

*4. Managing DAO layer*

*5. Page navigation*

*What is a servlet ?*

*It is a Java class (without main method)*

*Represents a dynamic web component , whose life cycle will be managed by WC*

*Life cycle methods - init,service,destroy*

*10. Servlet API details --*

*refer to diag servlet-api.png*

*Objective - Test basic servlet life cycle -- init , service ,destroy*

*Creating & deploying HelloWorld Servlet.*

*Deployment of the servlet*

*1. Via annotation*

*eg : @WebServlet(value="/hello")*

*public class HelloWorldServlet extends HttpServlet {....}*

*Meaning : A class level annotation , for WC, to add the mapping between incoming url-pattern n the servlet.*

*A URL mapping's Map , will be created by WC , at the web app deployment time*

*key -- url-pattern eg : /hello*

*value -- Fully qualified servlet cls name (eg : pages.HelloWorldServlet)*

*URL :* [*http://host:port/day1.1/hello*](http://host:port/day1.1/hello)

*URI : /day1.1/hello*

*URL pattern : /hello*

*OR*

*2. Legacy approach to deploy a servlet w/o annotations : Use XML tags in web.xml*

*<servlet>*

*<servlet-name>abc</servlet-name>*

*<servlet-class>pages.SecondServlet</servlet-class>*

*</servlet>*

*<servlet-mapping>*

*<servlet-name>abc</servlet-name>*

*<url-pattern>/test2</url-pattern>*

*</servlet-mapping>*

*WC created map will have a*

*key : /test2*

*value : pages.SecondServlet*

*At the time of web app deployment , WC tries to populate URL map, from XML tags or using @WebServlet annotation*

*(via Reflection)*

*Objective 2: Test basic servlet life cycle -- init , service ,destroy (deployed via xml)*

*10. How to read request params sent from the client ?*

*javax.servlet.ServletRequest i/f methods*

*1. public String getParameter(String paramName)*

*2. public String[] getParameterValues(String paramName)*

*What is default loading policy of WC for servlets ?*

*Lazy*

*Meaning - WC will start life cycle of the servlet , only after client sends the 1st request to the servlet*

*Can you change it to eager ? Yes*

*Use Case : Typically for time consuming(heavy weight) initializations*

*eg : setting up DB conn, setting up spring frmwork....*

*HOW ?*

*@WebServlet (value="/test",loadOnStartup=1)*

*OR*

*use xml tag :*

*<load-on-startup>1</load-on-startup>*

*Solve -*

*Load HelloWorldServlet eagerly @ app deployment time.*

*Let HelloWorldServlet2 - get loaded by default in lazy manner.*