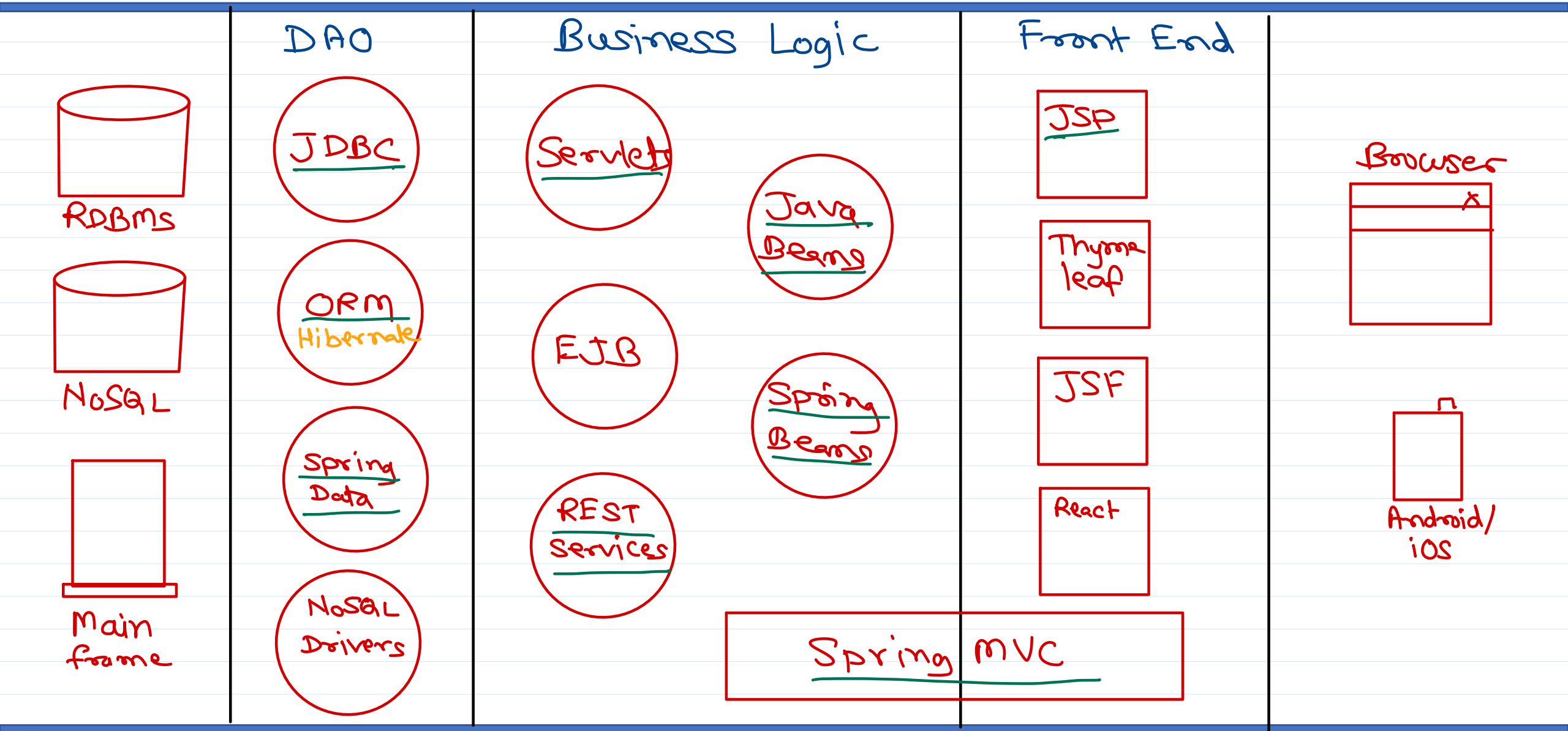
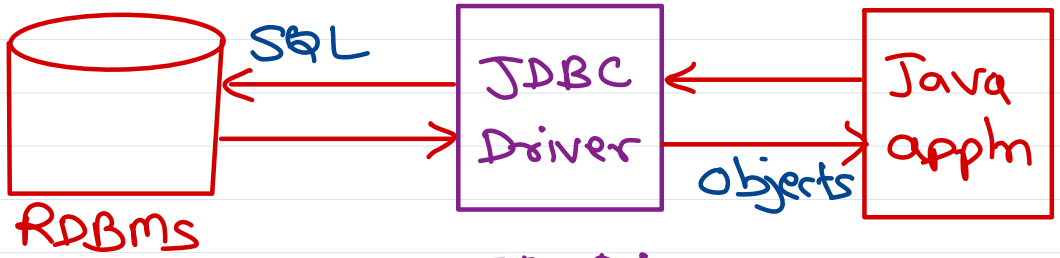
Advanced Java

Trainer: Nilesh Ghule

Java EE - Enterprise applications



JDBC



JDBC is a specification / standard.

JDBC spec = interfaces + helper classes.

1. Driver
2. Connection
3. Statement
4. ResultSet
5. DriverManager*
6. PreparedStatement
7. CallableStatement

} java.sql Package

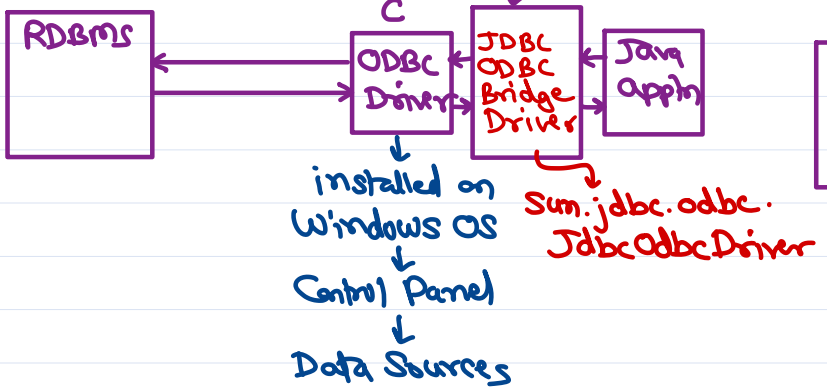
JDBC implementations = JDBC drivers.

- Diff drivers for diff RDBMS.
- given in form of .jar files = zip *.class files.
- classes in jdbc driver jar implement JDBC intfs.

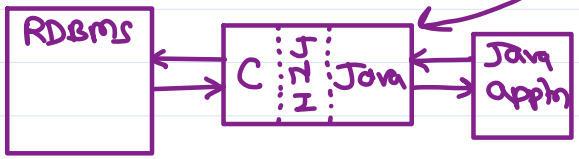
Type - IV Driver

- Fully in Java
- Platform independent
- Fast execution
- Db Specific

Type-I driver

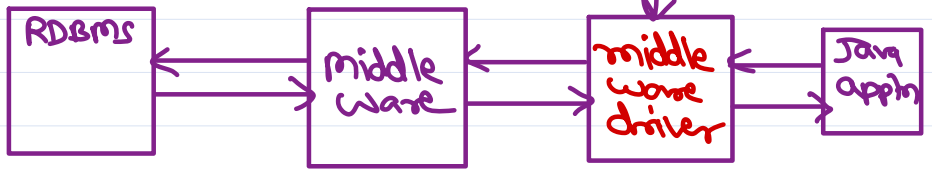


Type - II Driver



- DB specific
- not platform independent
- faster

Type - III Driver



- Specialized requirements only

JDBC Programming Steps - PreparedStatement.

① Add JDBC driver jar into project/classpath.

Eclipse - Java Project → Properties → Java Build Path → Libraries → Add External Jars → select mysql driver jar → Apply & Close

② Load JDBC driver & register it.

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

③ Create database connection.

```
url = "jdbc:mysql://localhost:3306/dbname";
con = DriverManager.getConnection(url, "user", "passwd");
```

④ Create parameterized Sql statement.

```
stmt = con.prepareStatement("param sql query");
```

⑤ Set params, execute query & process result.

```
stmt.setInt(1, val1);
stmt.setString(2, val2);
```

```
cnt = stmt.executeUpdate();
```

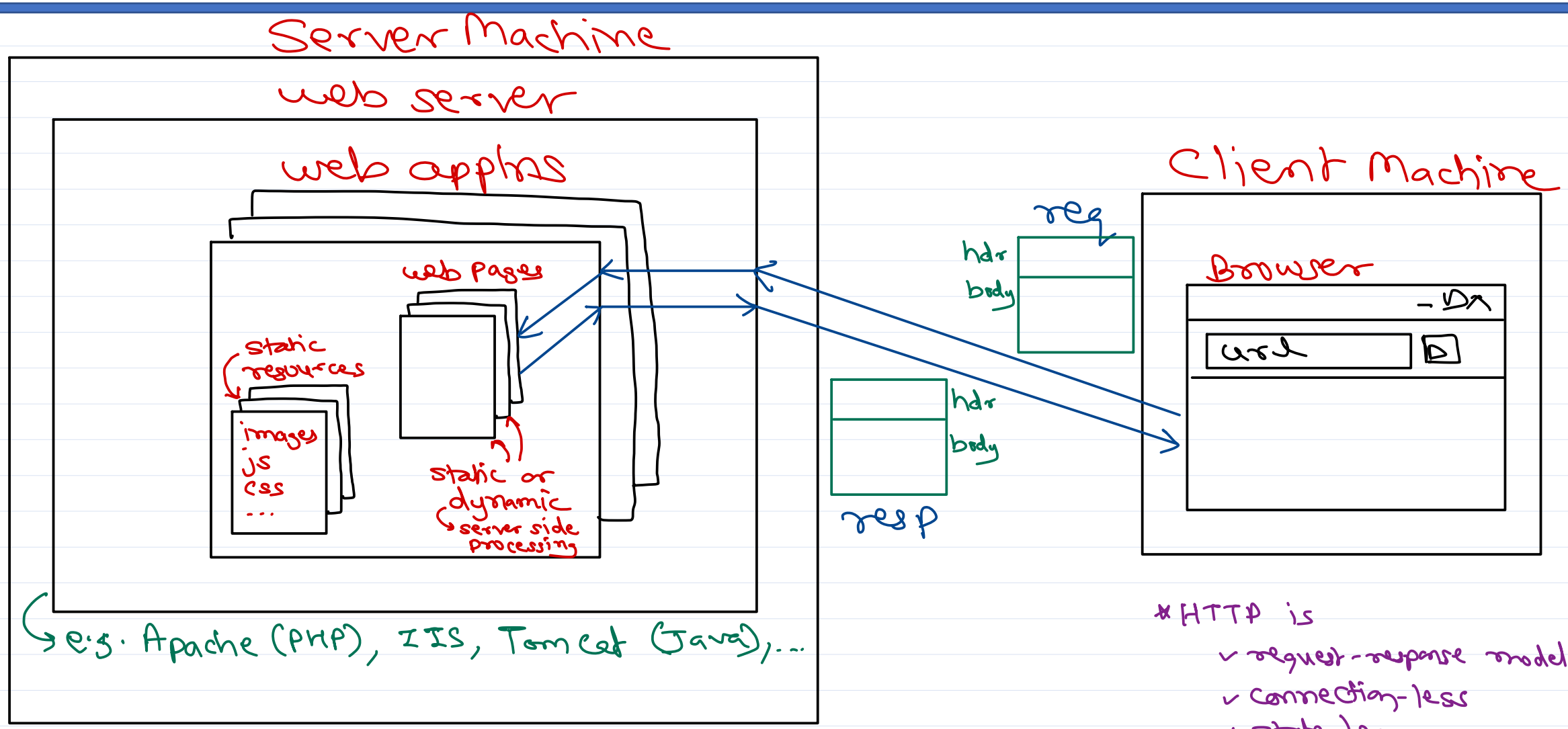
OR

```
rs = stmt.executeQuery();
while(rs.next()) {
    v1 = rs.getInt("col1");
    v2 = rs.getString("col2");
    ...
}
rs.close();
```

⑥ close statement & Connection.

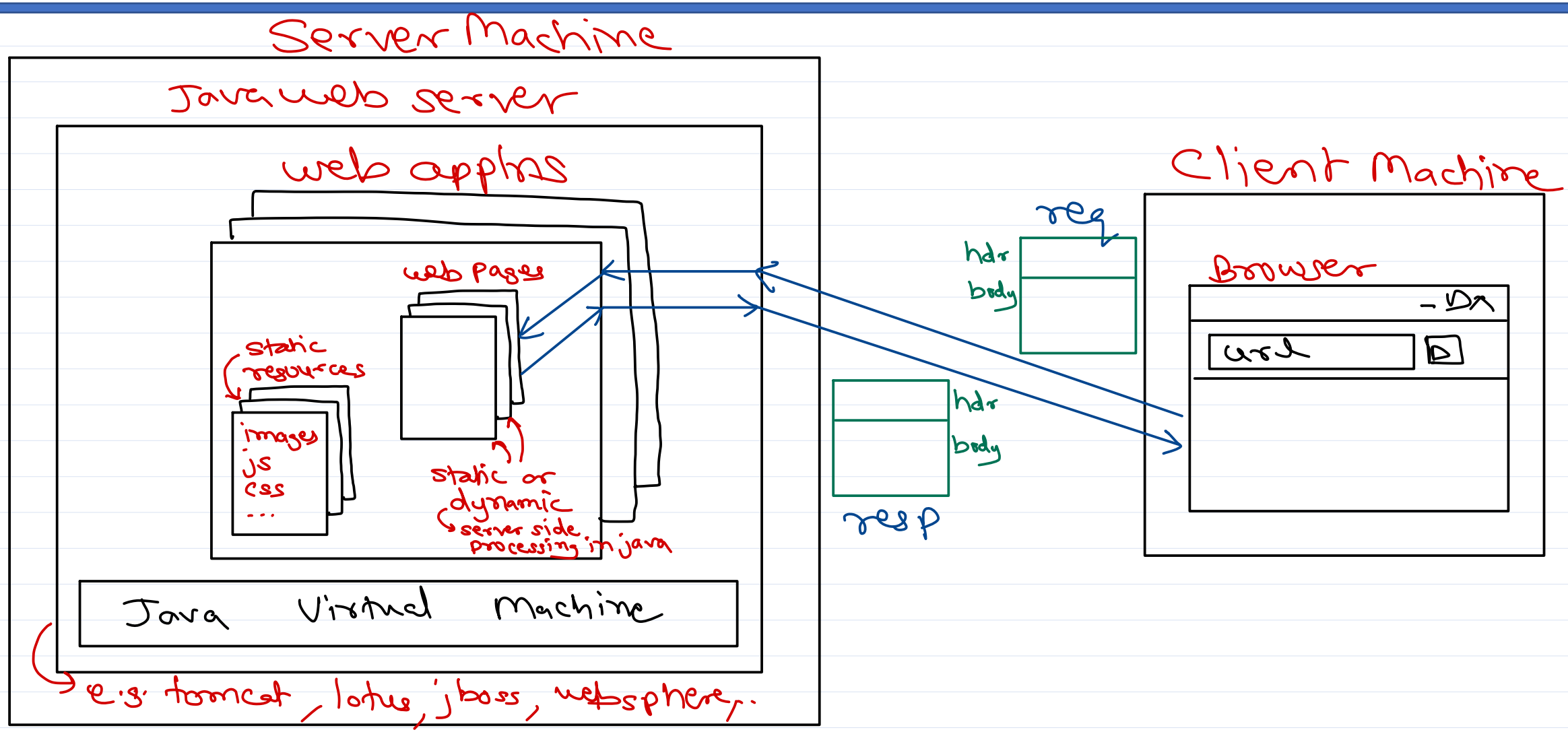
```
stmt.close();
con.close();
```

HTTP Protocol



* HTTP is
  - ✓ request-response model
  - ✓ connection-less
  - ✓ state-less

Java Web Server



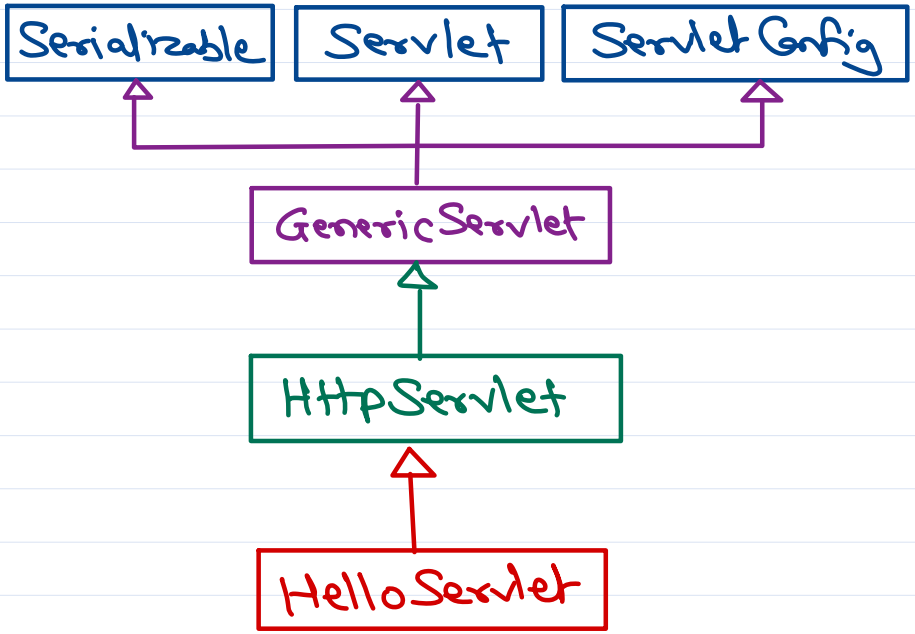
Java servers

Sun Microsystem
- standards / specifications.
  - JVM
  - JDBC
  - Java EE ← to be followed by java web servers & java web appln.
    - Servlet
    - JSP
    - JSF
    - JPA
    - JTA
    - EJB
    - JMS
    - ...

Java Servers
- web server = web container + extra services.
  - web container = servlet engine + JSP engine
    - e.g. tomcat, jetty, lotus, ...
  - extra services.
    1. https (SSL)
    2. connection pooling
    3. JNDI
    4. ...
- application server = web server + ejb container
  - web container + extra services.
    - web container = servlet engine + JSP engine
    - extra services.
      1. https (SSL)
      2. connection pooling
      3. JNDI
      4. ...
  - ejb container → scalable business logic components
  - e.g. jboss, websphere, weblogic, ...

Apache Tomcat → default port = 8080
- bin → tomcat bins, startup/shutdown.
- lib → tomcat libs (jars)
- conf → tomcat-users.xml, server.xml (change port, ssl, ...)
- webapps → hot deployment folder
- logs → log files (monitoring)
- work → JSP processing
- temp → temp files.

Servlet hierarchy




```
class HelloServlet extends HttpServlet {
    public void init(cfg) {
        // one time initialization
    }
    public void doGet(req, resp) {
        // get req & produce response
    }
    public void destroy() {
        // deinitialization
    }
}
```

Servlet interface:

1. void init(ServletConfig cfg) throws ServletException;
2. void service(ServletRequest req, ServletResponse resp) throws ServletException, IOException;
3. void destroy();

GenericServlet abstract class:

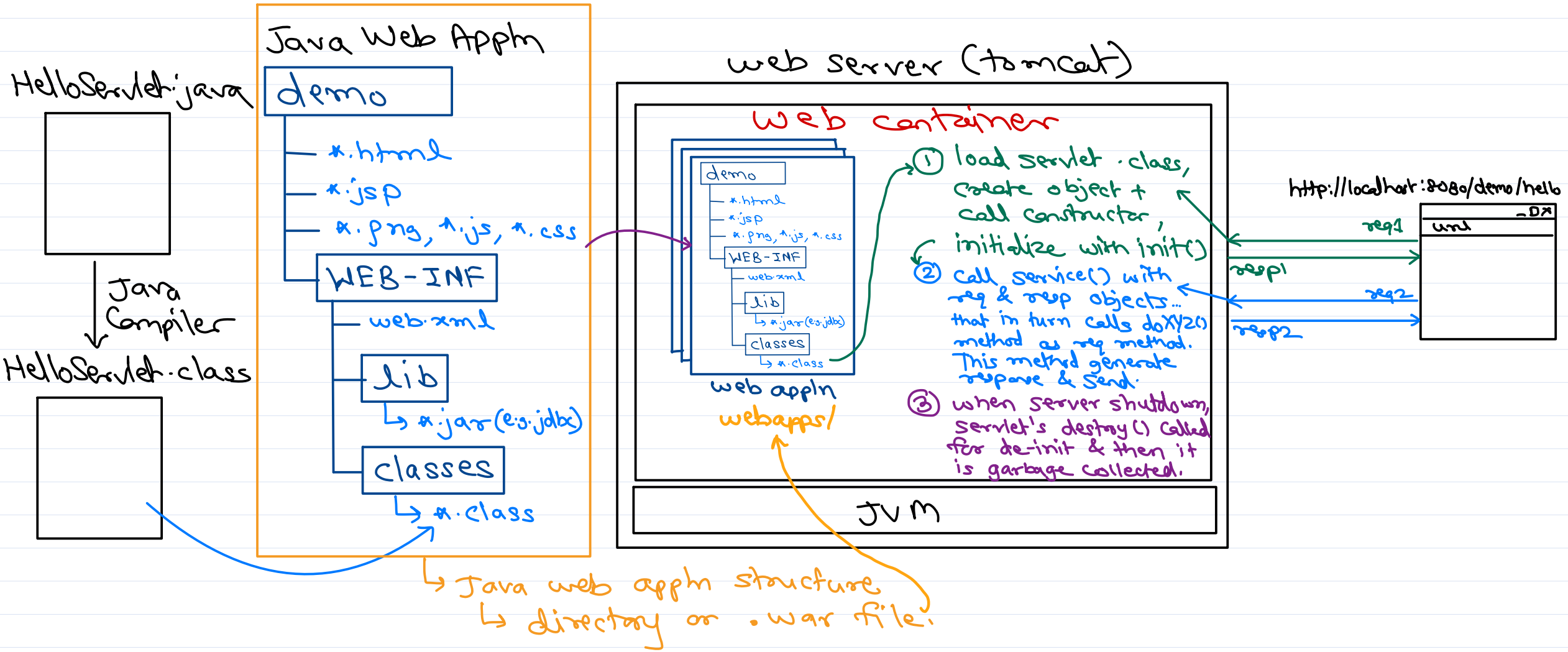
1. void init(ServletConfig cfg) {...}
2. void service(ServletRequest req, ServletResponse resp); — abstract
3. void destroy() {...}

HttpServlet class:

```
2. void service(req, resp) {
    method = req.getMethod();
    if (method.equals("GET"))
        doGet(req, resp);
    if (method.equals("POST"))
        doPost(req, resp);
    ...
}
```

4. doGet() {...}
5. doPost() {...}

HelloServlet execution



Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>