	Assignment Rubrics	Marks split up	Marks Score	Total Marks
Q1	Explanation of JDBC.	~~20M~~ 5M		
	Connection pooling	6M		
	SQL Queries	5M		
	Statement Types	4M		
Q2	Lifecycle phases Explanation	6M		
	Embedding Java code	5M		
	Advantages & Disadvantages	5M		
	clarity & Depth.	4M		
Q3	Code Implementation	8M		
	HTML Table structure	5M		
	Alternating colors logic	4M		
	explanation	3M		
Q4	Code Implementation	8M		
	Pattern Extraction	5M		
	XML file Generation	4M		
	DTD vs XML Schema Comparison	3M		

Assignment - 4

1. Why JDBC is Essential in Building Database-Driven Applications:

JDBC is essential because it provides a standard API for Java applications to interact with databases.

Achieving JDBC connection pooling using JDBC Data source and JNDI in Apache Tomcat.

Configure Data Source

```
<Resource  name = "jdbc/MyDB"
           auth = "container"
           type = "javax.sql.Datasource"
           maxtotal = "20"
           maxIdle = "10"
           Max wait millis = "10000"
           username = "dbuser"
           passwdd = "dbpasswdd"
           driver class Name = "com.mysql.cj.jdbc.Driver"
           url = "jdbc:mysql://localhost:3306/my
                                    database"/>
```

Lookup Data source in Javacode using JNDB

```
import javax.naming.context;
import javax.naming.InitialContext;
import javax.sql.Datasource;
import javax.sql.connection;
public class Databaseutil {
        public static Connection getconnection() throws
                                        exception {

        Context initcontext = new InitialContext();
        Datasource ds = (Datasource) initcontext lookup
```

```
("java:/comp/env/jdbc/myDB");
        return ds.getConnection();
    }
}
```

Executing SQL Queries using JDBC statement:

① using a statement:

```
try(Connection conn = DatabaseUtil.getConnection();
    Statement stmt = conn.createStatement()){
    String query = "SELECT * FROM users";
    ResultSet rs = stmt.executeQuery(query);
    while(rs.next()){
        System.out.println("User ID:" + rs.getInt("id") + " " +
                           "Name:" + rs.getString("name"));
    }
}
```

② Using a prepared Statement:

```
try(Connection conn = DatabaseUtil.getConnection();
    PreparedStatement pstmt = conn.prepareStatement
        ("SELECT * FROM users WHERE id = ?")){
    pstmt.setInt(1,1);
    ResultSet rs = pstmt.executeQuery();
    while(rs.next()){
        System.out.printf("User ID: " + rs.getInt("id") + ",
                          "Name: " + rs.getString("name"));
```

③ Using a Callable statement for stored procedures:

```
try(Connection conn = DatabaseUtil.getConnection();
    CallableStatement cstmt = conn.prepareCall("{
        call getUserById(?)}")){
    cstmt.setInt(1,1);
```

```
ResultSet rs = cstmt.executeQuery();
while (rs.next()){
    System.out.println("User ID:" + rs.getInt(":id")
        + ", Name:" + rs.getString("name"));

}
}
```

Output:

statement Example o/p:

User ID: 1, Name: Janani

User ID: 2, Name: Indhu

Prepared statement:

User ID: 1, Name: Janani

Collable statement:

User ID: 1, Name: Vels Janani

2. Life cycle phases of a JSP page

1. Translation page phase
2. Compilation page phase
3. Initialization phase
4. Request processing phase
5. Destruction phase.

Embedding Java Code in JSP

1. Scriplets:

```
<%. int sum = 5+10; %.>
<P> The sum is: <%. =sum %.>
```

Output:

The sum is: 15

2. Declaration

<%! int add (int a, int b) { return a+b; } %>

<P> The result is : <%= add (3,7) %> </P>

Output :

The result is : 10

3. Expressions :

<P> Current time : <%= new java.util.Date() %> </P>

Output :

Current time : Mon Sep 09 09:30:00 PDT 2024.

Scriplets :

Advantages : Easy to use for embedding simple Java logic.

Disadvantages : Leads to messy code, difficult to maintain.

Declarations :

Advantages : Useful for declaring reusable methods and variables across multiple requests.

Disadvantages : Can clutter JSP with Java code, leading to poor seperation of concerns.

Expressions :

Advantages : Simplifies outputting dynamic content directly in JSP.

Disadvantages : Limited to expressions.

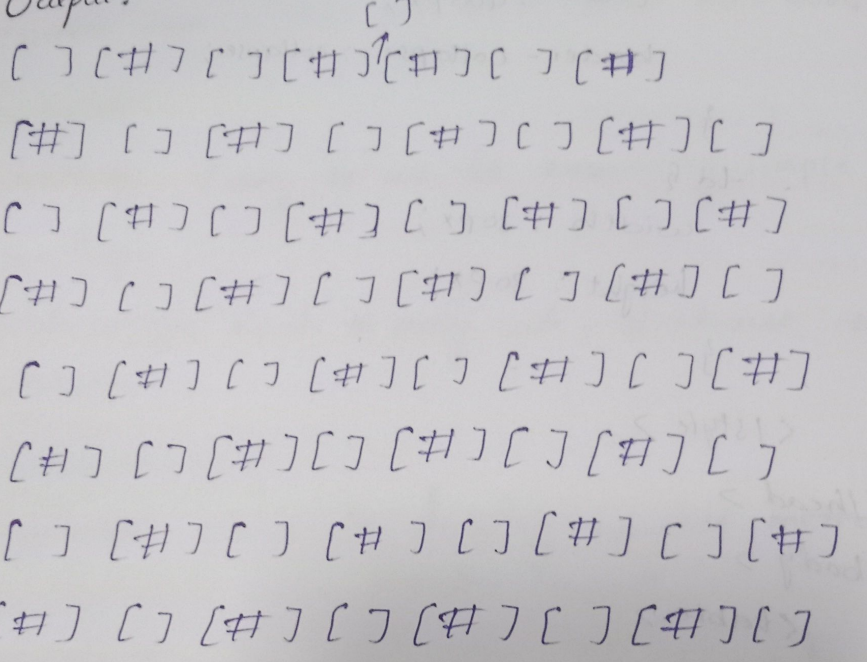
(3) Generates a Chessboard using HTML tables. Width of 400px (total) and each cell have a hight & width of 30px.

PHP code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>chessboard</title>
    <style>
        table {
            width: 400px;
            border-collapse: collapse;
        }
        td {
            width: 30px;
            height: 30px;
        }
    </style>
</head>
<body>
    <table>
        <?php
        // Loop for 8 rows
        for ($row = 0; $row < 8; $row++) {
        echo "<tr>";
        for ($col = 0; $col < 8; $col++) {
          if (($row + $col) % 2 == 0) {
            echo "<td style="background-color:
                white;"></td>";
          } else {
            echo "<td style='background-color:
                black;'></td>";
          }
```

```
        }
            echo "</tr>";
        }
        ?>
    </table>
</body>
</html>
```

Output:



④ PHP Applications to Extract Data and store in XML.

Steps:

1. Read content from a Text file.
2. Extract patterns using Regular
3. Create and store Results in an XML file
4. Define XML schema.

PHP code:

```
<? PHP
$ file name : 'input. txt';
$ content = file-get- contents ($-filename);
preg - match - all (/ [ a-zA-Z0-9. - %. + - ] + @ [a-zA-Z
                                    0-9. - ] + \. [a-zA-Z]{2,}
                                    /', $ content, $ emails );
preg - match - all ('/ \+ ? [0-9]{1,3} [ - .\s ] ? [0-9]{1,4}
                    [ - .\s ] ? [0-9]{3,4} /', $ content, $ phones);
$ xml = new Simple XML Element ('<data></data>');
$ email Element = $ XML -> add child ('emails');
for each ($ emails [0] as $ email){
    $ email Element -> add child ('email', $ email);
}
$ phone Element = $ XML -> add child ('phones');
for each ($ phones [0] as $ phone){
    $ phone Element -> add child ('phone', $ phone);
}
$ XML -> as XML ('output. XML);
echo " Data extracted and saved to output. XML "
?>
```

Output:

```
<data>
    <emails>
    <email> example1@example.com </email>
    <email> example2@example.com </email>
    </email>
  <phones>
      <phone> +123-456-7890 </phone>
      <phone> 987-654-3210 </phone>
  </phones>
</data.
```