1. **JSON:**

Owner = {

"name":"Test",

"age":30,

"cars": {

"car1":"Honda",

"car2":"Corolla",

"car3":"cultus"

}

}

The above code shows a nested JSON object the cars object is encapsulated in the owner object.

1. **Android http url connection:**

URL url;

HttpURLConnection urlConnection = null;

try {

url = new URL("http://www.mysite.se/index.asp?data=99");

urlConnection = (HttpURLConnection) url

.openConnection();

InputStream in = urlConnection.getInputStream();

InputStreamReader isw = new InputStreamReader(in);

int data = isw.read();

while (data != -1) {

char current = (char) data;

data = isw.read();

System.out.print(current);

}

} catch (Exception e) {

e.printStackTrace();

} finally {

if (urlConnection != null) {

urlConnection.disconnect();

}

}

The code above shows the http url connection for android. In this the connection is opened using the url of a particular web page to get data, after the connection is established the data is read using the input reader object and used as desired. The connection is closed after getting the required data

1. **Servlet:**

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

/\*\*

\* Servlet implementation class MyServlet

\*/

@WebServlet("/MyServlet")

public class MyServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

/\*\*

\* @see HttpServlet#HttpServlet()

\*/

public MyServlet() {

super();

// TODO Auto-generated constructor stub

}

/\*\*

\* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)

\*/

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// TODO Auto-generated method stub

// Set response content type

String message = "Welcome to the Servlet";

response.setContentType("text/html");

// Actual logic goes here.

PrintWriter out = response.getWriter();

out.println("<h1>" + message + "</h1>");

}

/\*\*

\* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)

\*/

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// TODO Auto-generated method stub

String message = "Servlet POST";

response.setContentType("text/html");

// Actual logic goes here.

PrintWriter out = response.getWriter();

out.println("<h1>" + message + "</h1>");

}

}

In the above code a java servlet is written named My Servlet. The doGet() and doPost() methods of the servlet are implemented, the message is printed as heading 1 in both of the methods

1. **Junit Testing:**

import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;

public class MyTests {

@Test

public void multiplicationOfZeroIntegersShouldReturnZero() {

MyClass tester = new MyClass(); // MyClass is tested

// assert statements

assertEquals(0, tester.multiply(10, 0), "10 x 0 must be 0");

assertEquals(0, tester.multiply(0, 10), "0 x 10 must be 0");

assertEquals(0, tester.multiply(0, 0), "0 x 0 must be 0");

}

}

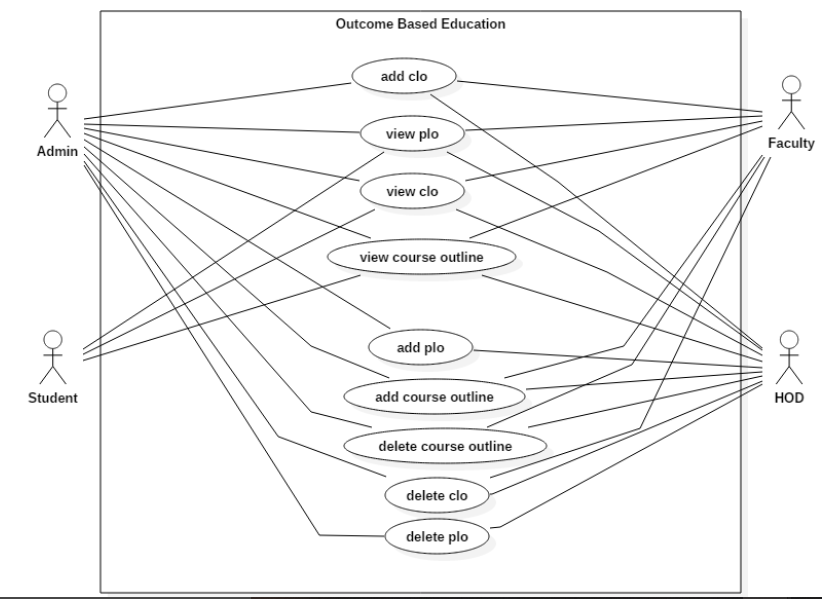
The above code demonstrates Junit testing in this code it tests MyClass for its multiply method that has two parameters of type integer.

1. **System Testing:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # |  |  |  |  |  |
| 1 |  |  |  |  |  |
| 2 |  |  |  |  |  |
| 3 |  |  |  |  |  |
| 4 |  |  |  |  |  |
| 5 |  |  |  |  |  |
| 6 |  |  |  |  |  |
| 7 |  |  |  |  |  |
| 8 |  |  |  |  |  |
| 9 |  |  |  |  |  |
| 10 |  |  |  |  |  |

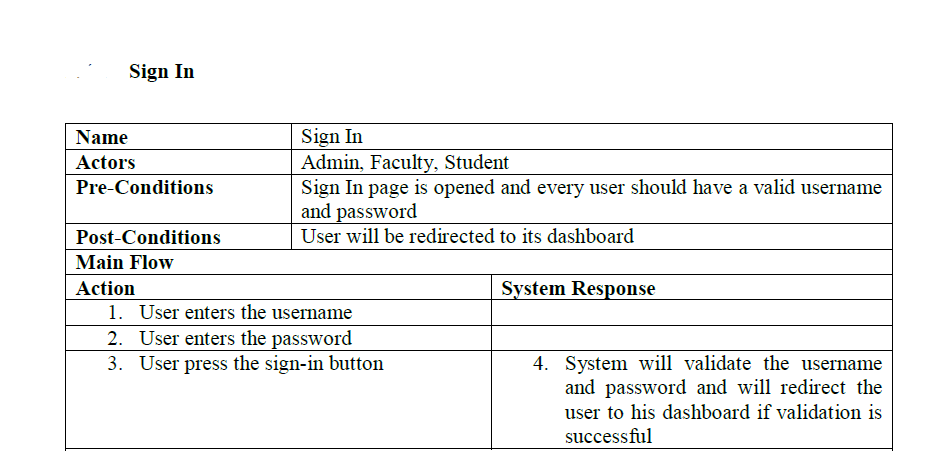
System testing is done to test the integrated system that whether it works as desired as an integrated system. This testing is done after the unit testing in which the individual units are tested for their functionality. System testing ensures that the individual units will not deviate from their functionality when they are integrated with other units and the system will perform in the right way after the overall integration.

1. **Use case Diagram:**

****

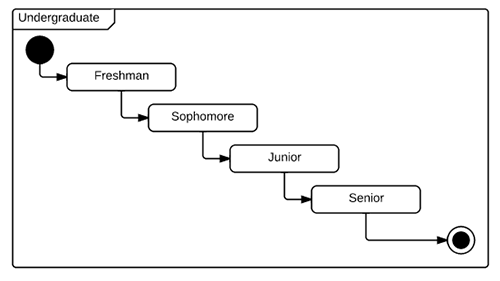
Use case diagram of an application to implement Outcome Based Education is shown above. In this diagram the use case of various actors of the system I.e. Student, faculty, admin and HOD are presented in the rectangle with each actor pointing to his use case.

1. **Use case Document:**

****

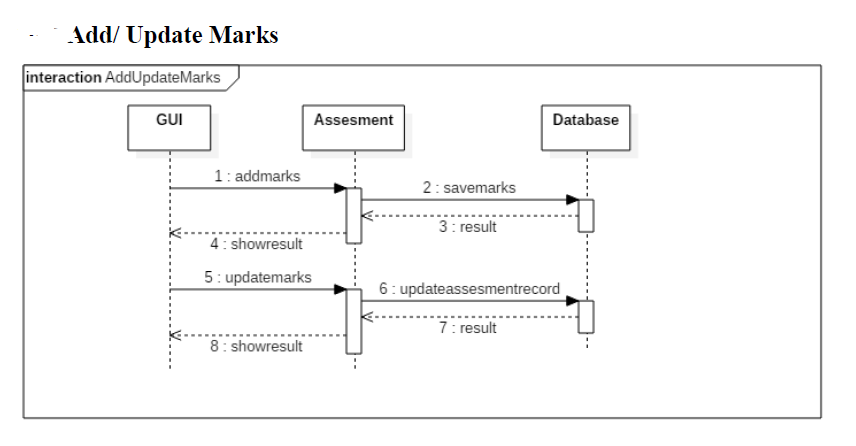
The sign in use case document has been shown above in which the user possible actions for the sign in has been listed and the corresponding system response against each action is also listed.

1. **State Diagram:**

****

The diagram above shows the different states of an undergraduate student. The filled black circle represents the start of state diagram and the black circle with the token marks the end of state diagram the intermediate states of the student are shown in the diagram

1. **Sequence Diagram:**

****

The sequence diagram of Add marks use case is shown above in which it depicts the entire process of adding marks. In this the user communicates with the GUI to add marks after he enters the marks, they are passed to the assessment class object and then recorded in the database for persistent storage.