Assignment No.: B2

1. TITLE

A Web application for Concurrent implementation of ODD-EVEN SORT is to be designed using Real Time Object Oriented Modelling (ROOM). Give the necessary design diagrams and write the test cases for the white box testing. Draw Concurrent Collaboration Diagrams.

2. PREREQUISITES

- 64-bit Fedora or equivalent OS with 64-bit Intel-i5/i7
- Java 1.7.0

3. OBJECTIVE

- Understand and Implement the Meaning of Odd-Even sort.
- Write test cases for White Box Testing

4. THEORY

In computing, an odd–even sort is a relatively simple sorting algorithm, developed originally for use on parallel processors with local interconnections. It is a comparison sort related to bubble sort, with which it shares many characteristics. It functions by comparing all odd/even indexed pairs of adjacent elements in the list and, if a pair is in the wrong order (the first is larger than the second) the elements are switched. The next step repeats this for even/odd indexed pairs (of adjacent elements). Then it alternates between odd/even and even/odd steps until the list is sorted.

Consider the concurrent odd-even transposition sort of an even-sized array. For 8 elements, 4 threads operate concurrently during the odd phase, and 3 threads operate concurrently during the even phase. Assume that threads, which execute during the odd phase, are named O1, O2, O3, and O4 and threads, which execute during the even phase, are named E1, E2 and E3. During an odd phase, each odd-phase thread compares and exchanges a pair of numbers. Thread O1 compares the first pair of numbers, thread O2 compares the second pair of numbers, thread O3 compares the third pair, and O4 compares the fourth pair of numbers. During an even phase, the first and last elements of the array are not processed; thread E1 compares the second and third array elements, thread E2 compares the fourth and fifth array elements, and thread E3 compares the sixth and seventh elements. The algorithm runs for a total of 8 phases (i.e., equal to the number of values in the data file).

In Odd Even Sort compare the Element available on Even Index sort and for Odd sort compare the Element available on Odd index. in this assignment JSP used for Web application development purpose the remaining logic for sorting purpose is bubble sort. JSP technology is used to create web application just like Servlet technology. It can be thought of as an extension to servlet because it provides more functionality than servlet such as expression language, jstl etc. A JSP page consists of HTML tags and JSP tags. The jsp pages are easier to maintain than servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tag etc

Advantage of JSP over Servlet
There are many advantages of JSP over servlet. They are as follows

1) Extension to Servlet
JSP technology is the extension to servlet technology. We can use all the features of servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.

2) Easy to maintain
JSP can be easily managed because we can easily separate our business logic with presentation logic. In servlet technology, we mix our business logic with the presentation logic.

3) Fast Development
No need to recompile and redeploy. If JSP page is modified, we don't need to recompile and redeploy the project. The servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

4) Less code than Servlet
In JSP, we can use a lot of tags such as action tags, jstl, custom tags etc. that reduces the code. Moreover, we can use EL, implicit objects etc.

Life cycle of a JSP Page
The JSP pages follows these phases:

- Translation of JSP Page.
- Compilation of JSP Page.
- Classloading (class file is loaded by the classloader).
- Instantiation (Object of the Generated Servlet is created).
- Initialization (jspInit() method is invoked by the container).
- Reqeust processing (_jspService() method is invoked by the container).
- Destroy (jspDestroy() method is invoked by the container).

Real-Time Object-Oriented Modeling (ROOM)
Model real time systems based on timeliness, dynamic internal structure, reactiveness, concurrency and distribution, using the ROOM notation. **ROOM** is an object-oriented methodology for real-time systems developed originally at Bell-Northern Research. ROOM is based upon a principle of using the same model for all phases of the development process. ROOM models are composed of *actors* which communicate with each other by sending messages along *protocols*. Actors may be hierarchically decomposed, and may have behaviors described by ROOM charts, a variant of Harel's state charts. Descriptions of actors, protocols, and behaviors can all be reused through inheritance.

White Box Testing
White box testing is a testing technique, that examines the program structure and derives test data from the program logic/code. White Box Testing (WBT) is also known as Code-Based Testing or Structural Testing. White box testing is the software testing method in which internal structure is being known to tester who is going to test the software. White box testing is also known as clear box testing, open box testing, logic driven testing or path driven testing or structural testing and glass box testing. The White-box testing is one of the best method to find out the errors in the software application in early stage of software development life cycle.

- **What do you verify in White Box Testing ?**

In the White box testing following steps are executed to test the software code:

- Basically verify the security holes in the code.
- Verify the broken or incomplete paths in the code.
- Verify the flow of structure mention in the specification document
- Verify the Expected outputs
- Verify the all conditional loops in the code to check the complete functionality of the application.
- Verify the line by line or Section by Section in the code & cover the 100% testing. Above steps can be executed at the each steps of the STLC i.e. Unit Testing, Integration & System testing.

- **White Box Testing Techniques**

Here are some white box testing techniques used in White Box Testing?

1. **Statement Coverage:**

In this white box testing technique try to cover 100% statement coverage of the code, it means while testing the every possible statement in the code is executed at least once.
Tools: To test the Statement Coverage the Cantata++ can be used as white box testing tool.

2. **Decision Coverage:**

In this white box testing technique try to cover 100% decision coverage of the code, it means while testing the every possible decision conditions like if-else, for loop and other conditional loops in the code is executed at least once.

Tools: To cover the Decision Coverage testing in the code the TCAT-PATH is used. This supports for the C, C++ and Java applications.

3. **Condition Coverage:**

In this white box testing technique try to cover 100% Condition coverage of the code, it means while testing the every possible conditions in the code is executed at least once.

4. **Decision/Condition Coverage:**

In this mixed type of white box testing technique try to cover 100% Decision/Condition coverage of the code, it means while testing the every possible Decisions/Conditions in the code is executed at least once.

5. **Multiple Condition Coverage:**

In this type of testing we use to cover each entry point of the system to be execute once.

In the actual development process developers are make use of the combination of techniques those are suitable for there software application.

How do you perform White Box Testing?

Let take a simple website application. The end user is simply access the website, Login & Logout, this is very simple and day 2 days life example. As end users point of view user able to access the website from GUI but inside there are lots of things going on to check the internal things are going right or not, the white box testing method is used. To explain this we have to divide this part in two steps. This is all is being done when the tester is testing the application using White box testing techniques.

STEP 1) UNDERSTAND OF THE SOURCE CODE
STEP 2) CREATE TEST CASES AND EXECUTE

STEP 1) UNDERSTAND OF THE SOURCE CODE

The first & very important steps is to understand the source code of the application is being test.

As tester should know about the internal structure of the code which helps to test the application. Better knowledge of Source code will helps to identify & write the important test case which causes the security issues & also helps to cover the 100% test coverage. Before doing this the tester should know the programming language what is being used in the developing the application. As security of application is primary objective of application so tester should aware of the security concerns of the project which help in testing. If the tester is aware of the security issues then he can prevents the security issues like attacks from hackers & users who are trying to inject the macious things in the application.

STEP 2) CREATE TEST CASES AND EXECUTE

In the second step involves the actual writing of test cases based on Statement/Decision/Condition/Branch coverage & actual execution of test cases to cover the 100% testing coverage of the software application. The Tester will write the test cases by diving the applications by Statement/Decision/Condition/Branch wise. Other than this tester can include the trial and error testing, manual testing and using the Software testing tools as we covered in the previous article.

- **Types of White Box Testing:**

1. Unit Testing 2. Integration Testing

1. Unit Testing
What is a Unit?
Unit is a single smallest independent component. Unit Testing is categorized as:
1. Execution Testing
2. Operations Testing
3. Mutation Testing

2. **Integration Testing:**

After making independent components, these components are joined together to check if they are working fine together as one application.
Example: Compose Mail and Sent Items. After composing and sending mails, mails are checked in sent items. Different modules should inter talk with each other.

 Approaches in Integration Testing:
1. Top Down Approach:

When main module is ready and sub module is under development. Main module is dependent on sub module. Here, stubs comes into picture. Stubs are temporary programs which takes the place of module under development. Ready module is tested using Stubs.

2. Bottom Up Approach:

When Sub module is ready and main module is under development. Here, driver are the temporary programs which are used to test sub modules.

3. Hybrid Approach:

It is the combination of Top Down and Bottom Up Approach.

5. MATHEMATICAL MODELS

Let, S be the System Such that,
A={ S, E, I,O, F, DD, NDD, success, failure }
Where,
S= Start state,
E= End State,
I= Set of Input
O= Set of Out put
F =Set of Function
DD=Deterministic Data
NDD=Non Deterministic Data
Success Case: It is the case when all the inputs are given by system are entered correctly. Failure Case: It is the case when the input does not match the validation Criteria.

6. CONCLUSION

A Web application is created for Concurrent implementation of ODD-EVEN SORT using Real time Object Oriented Modeling (ROOM).

```
File index.jsp
<%--
   Document   : index
   Created on : 9 Mar, 2016, 11:26:49 AM
   Author     : @@
--%>

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
   <head>
       <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
       <title>ODD EVEN SORT</title>
   </head>
   <body>
       <h1>ODD EVEN SORT</h1>
       <form action="Oddevents">
           Enter number 1
           <input type="text" name="no0" value="" /><br><br>
           Enter number 2
           <input type="text" name="no1" value="" /><br><br>
           Enter number 3
           <input type="text" name="no2" value="" /><br><br>
           Enter number 4
           <input type="text" name="no3" value="" /><br><br>
           Enter number 5
           <input type="text" name="no4" value="" /><br><br>
           Enter number 6
           <input type="text" name="no5" value="" /><br><br>
           <input type="submit" value="Display Answer" name="btn"/><br><br>
           <input type="reset" value="Reset Value" /><br><br>
       </form>
   </body>
</html>
```

File: Oddevents.java

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

public class Oddevents extends HttpServlet {

   protected void processRequest(HttpServletRequest request, HttpServletResponse response)
           throws ServletException, IOException {
       response.setContentType("text/html;charset=UTF-8");
       PrintWriter out = response.getWriter();
       int i, n;
       int[] array = new int[6];
       try {

           out.println("<!DOCTYPE html>");
           out.println("<html>");
           out.println("<head>");
           out.println("<title>Servlet Oddevents</title>");
```

```
            out.println("</head>");
            out.println("<body>");

            for(i=0;i<6;i++)
            {
                String num=request.getParameter("no"+i);
                n=Integer.parseInt(num);
                array[i]=n;
            }
            out.println("Values Before the sort:\n\n");
            for (i = 0; i < array.length; i++) {
                out.println(array[i] + " \n ");

            }
            out.println();
            odd_even_srt(array, array.length);
            out.println("Values after the sort:\n\n");
            for (i = 0; i < array.length; i++) {
                out.println(array[i] + " \n ");
            }
            out.println();

            out.println("</body>");
            out.println("</html>");
        } finally {
            out.close();
        }
    }

     public static void odd_even_srt(int array[], int n) {
        for (int i = 0; i < n / 2; i++) {
            for (int j = 0; j + 1 < n; j += 2) {
                if (array[j] > array[j + 1]) {
                    int T = array[j];
                    array[j] = array[j + 1];
                    array[j + 1] = T;
                }
            }
            for (int j = 1; j + 1 < array.length; j += 2) {
                if (array[j] > array[j + 1]) {
                    int T = array[j];
                    array[j] = array[j + 1];
                    array[j + 1] = T;
                }
            }
        }
    }

    // <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click on the + sign on
the left to edit the code.">
    /**
     * Handles the HTTP <code>GET</code> method.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
     * Handles the HTTP <code>POST</code> method.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
     * Returns a short description of the servlet.
     *
     * @return a String containing servlet description
     */
    @Override
    public String getServletInfo() {
        return "Short description";
    }// </editor-fold>

}
```

File: mytest2.java

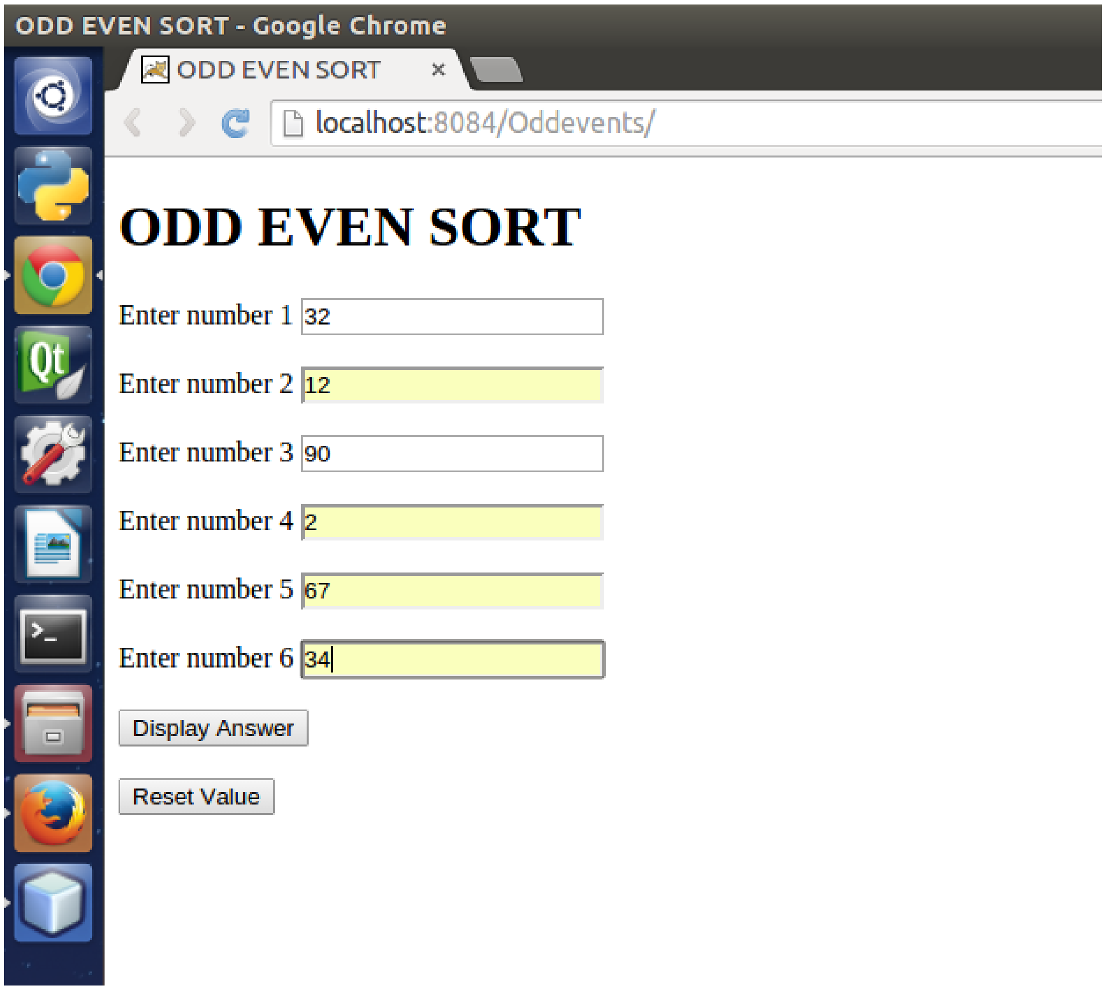
```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;

public class mytest2 {

    public static void main(String[] args) {

        WebDriver driver = new FirefoxDriver();
        String baseUrl = "http://localhost:8084/OddEvenSort/";
        driver.get(baseUrl);
        String expected = "JSP Page";
        String actual = "";
        driver.manage().window().maximize();
        actual = driver.getTitle();
        if (actual.equals(expected)) {
            System.out.println("Title test passed");
        } else {
            System.out.println("Title test failed");}
            WebElement text=driver.findElement(By.name("no0"));
            text.sendKeys("5");
            WebElement text1=driver.findElement(By.name("no1"));
            text1.sendKeys("25");
            WebElement text2=driver.findElement(By.name("no2"));
            text2.sendKeys("50");
            WebElement text3=driver.findElement(By.name("no3"));
            text3.sendKeys("56");
            WebElement text4=driver.findElement(By.name("no4"));
            text4.sendKeys("23");
            WebElement text5=driver.findElement(By.name("no5"));
            text5.sendKeys("90");
            WebElement btn=driver.findElement(By.name("btn"));
            btn.click();
            System.out.println(" test script sucessful");
            driver.close();

    }
}
```



ODD EVEN SORT

Enter number 1 32

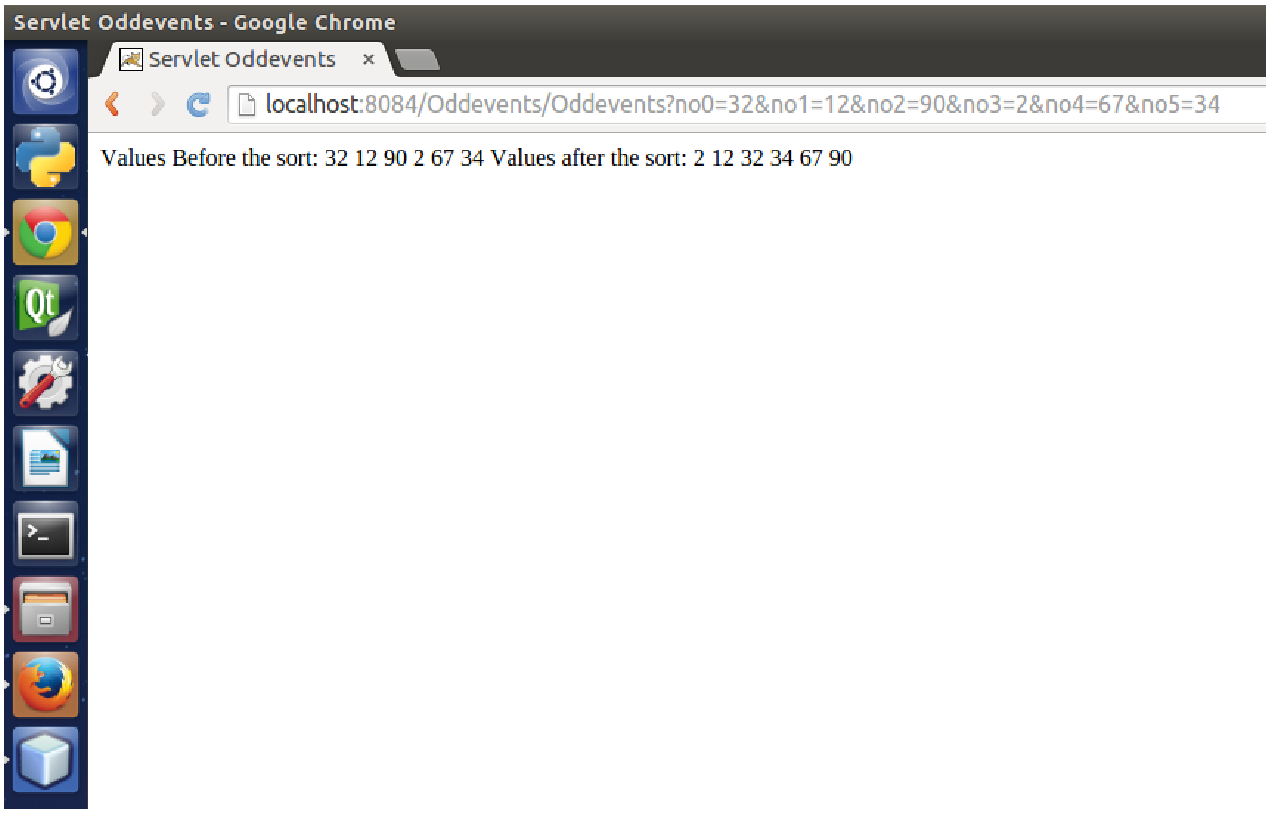
Enter number 2 12

Enter number 3 90

Enter number 4 2

Enter number 5 67

Enter number 6 34



```
localhost:8084/Oddevents/Oddevents?no0=32&no1=12&no2=90&no3=2&no4=67&no5=34
```

```
Values Before the sort: 32 12 90 2 67 34 Values after the sort: 2 12 32 34 67 90
```