2004

Olympiad delegation registration system

Xuetao Wang

OLYMPIAD DELEGATION REGISTRATION SYSTEM

A Project

Presented to the

Faculty of

California State University,

San Bernardino

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

---

by

Xuetao Wang

June 2004

OLYMPIAD DELEGATION REGISTRATION SYSTEM

A Project

Presented to the

Faculty of

California State University,

San Bernardino

---

by

Xuetao Wang

June 2004

Approved by:

[redacted]

Dr. David Turner, Chair, Computer Science

6/4/2004

Date

[redacted]

Dr. George Georgiou

[redacted]

Dr. Keith Schubert

ABSTRACT

"Olympiad Delegations Registration System" (ODRS) is a piece of server software to manage the registration of national delegations to the Olympiad over the Web. There are two types of users of the system: staff of the various National Olympic Committees (NOC), and staff of the central Organizing Committee for the Olympic Games (OCOG). According to the Olympic charter, each NOC must register its athletes and officials for participation in the Olympiad, and the OCOG verifies the qualification of these registrations. ODRS supplies several interfaces for NOCs and OCOG users to manage this registration process. The NOCs can add, modify, delete and automatically check the qualifications related to the athletes and officials in their own delegations, while OCOG can view all NOC registrations dynamically. Furthermore, ODRS may be integrated into a larger Olympic administration system with broader scope.

ACKNOWLEDGMENTS

I would like to thank the faculty of Computer Science department for giving me an opportunity to pursue my M.S. in Computer Science at California State University, San Bernardino. I owe a special appreciation for my graduate advisor, Dr. David Turner, who directed me through this entire effort to eventually accomplish my master's project, and other committee members, Dr. George Georgiou and Dr. Keith Schubert who gave me their valuable input. Meanwhile, a special thanks to the Department staff members Ms. Monica Gonzales and Mr. Kwon Soo Han. Both of them gave me much help during my pursuit for the degree at California State University, San Bernardino. Also, the support of the National Science Foundation under the award 9810708 is gratefully acknowledged.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

The Olympiad (short for Summer Olympic Games) is the world's largest sporting event in terms of the number of sports in the program, the number of athletes presented, and the number of different countries united in a single geographical area. It takes place every four years. From the date of the first games in 1896, it has been held twenty-seven times. The next two Olympiads (28^{th} and 29^{th}) will be held separately in 2004 at Athens and in 2008 at Beijing.

The International Olympic Committee (IOC) is the supreme authority of the Olympiad, which functions to ensure the regular celebration of the Olympic Games. IOC entrusts the organization of the Olympic Games to the National Olympic Committee (NOC) of the host country and its host city. The host NOC forms an Organizing Committee for the Olympic Games (OCOG), which communicates directly with the IOC. In the meantime, the IOC determines some International Sports Federations (ISFs) to administer one or several sports in the Olympiad. The relationship among these sport organizations during the Olympiad is demonstrated in the hierarchy illustration in Appendix A.

To be eligible for participating in the Olympiad, all country and region competitors must register through the NOC to which they belong. An authorized department of OCOG is responsible for all NOC delegation registration. "Olympiad Delegations Registration System" (ODRS) is a piece of server software that allows the NOC of participant countries to register their delegations of athletes and officials for participation the Olympiad through the Web. According to the Olympiad charter, each NOC must register on the Olympiad for its athletes, and OCOG verify the qualification of these registrations. ODRS supplies several interfaces for NOCs and OCOG users to manage these registrations. NOCs can add, modify delete and atomically check the qualification related to the athletes and officials in their own delegations, while OCOG can view all NOC registrations dynamically. And furthermore, ODRS may be integrated or implanted into super scale Olympic administration system

1.1 Purpose of this Project

The purpose of this project is to design, build and implement a web application system for the Olympiad delegation registration. All the pages and user registration information will be stored in a PostgreSQL database and retrieved by JAVA Servlet and JDBC. The main purpose of this

project is to provide an easy-to-register and web-base communication environment for the NOCs and the OCOG. Moreover, the system offers the authorization function to make sure that the pages are secure from malicious access. In the system, all the NOC users can manage their own registred information such as changing passwords, add or delete athletes and officials. Furthermore, in order to help OCOG staff approve the qualification of each NOC delegation, ODRS can atomically check whether those inputed information is correct or not in light of the IOC rules guiding Olympiad delegation registrations.

1.2 Project Products

This project leads to the following products:

- Implementation of ODRS: a working web site with JSP programs, Java programs and PostgreSQL database, which would achieve the needs of a communication board of a regular class. All the pages will be watched by the security system in order to keep the information on the pages correctly and even securely.
- Users manual: an implementation manual will be available for the users.
- Systems Manual: a project report (this report) will be available with design details and specifications.

CHAPTER TWO

OLYMPIAD DELEGATION REGISTRATION SYSTEM ARCHITECTURE

Olympiad delegation registration system (ODRS) implements a web system to provide an environment for the National Olympic Committees (NOCs) and the Organizing Committee of the Olympic Games (OCOG) to manage the registration of national delegations. Each NOC is authorized to manage the registrations for it country's delegation, while the OCOG is authorized to manage registration data from all NOCs. The components to implement the ODRS system are: a database server, a web server, graphical user interface components, and a database interface to programmatically access the database. The following figure describes the interaction among the components used in ODRS.

The components used to build ODRS were chosen with the following criteria: (i) the components should be shareware, i.e., available freely for non-commercial purposes, (ii) be part of a standard, i.e., do not depend on a specific operating system and hence are easily portable across systems with ease, (iii) database server independent, so that new and different versions of the server can be plugged in easily.

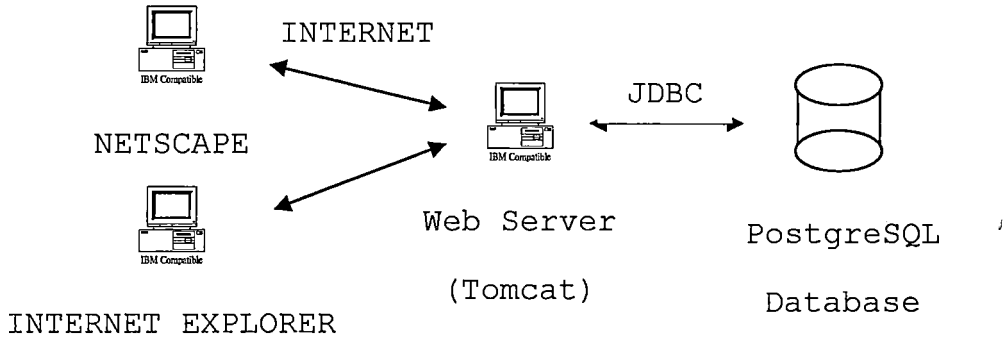

Figure 1. Olympiad Delegation Registration System Architecture

The user interface components are built by using HTML 4.01 forms, frames and Javascript. And the applications are launched using JavaServer Pages (JSP) and Java Servlet. JSP was used because it runs in the Tomcat Web server, which can be installed under Windows or Linux. Also, it is easy to process whole user input from the HTML forms. The reason that Java Servlets were used is that it has the advantages of portability and efficiency. By using Java Servlets, the Servlet can be executed in any other web server, which is the special property of Java Servlets, "write once, serve anywhere." Moreover, Java provides a generic API, the Java Database Connector (JDBC), to connect to databases.

The database used for ODRS is PostgreSQL. PostgreSQL is a real multi-user database and is royalty-free open source software. To use it, simply activate PostgreSQL in Linux.

Also, the availability of the JDBC driver for PostgreSQL is another important reason to choose it. Moreover, the same code could be used to link with another database by changing proper JDBC driver, thereby making it database independent.

2.1 Software Interfaces

- Internet browser: Netscape or Internet Explorer.
- Operating system: Unix/Linux.
- Database: PostgreSQL.
- Compiler: JDK 1.4.2.
- Language: HTML / JAVA / ANT / JSP.
- Database connector: JDBC.
- JSP Container/Web server: Jakarta Tomcat.

CHAPTER THREE

DATABASE DESIGN

3.1 Data Analysis

The design and implementation of the database schema depends on the properties of ODRS data. ODRS data consists of userid, password, ID, noc, event, familyname, givenname, gender, and birthday. In addition to the above general data, athletes for each event have unique data requirements. ODRS mainly consists of two kinds of registrations: officials and athletes. The data for officials include ID, noc, familyname, givenname, birthday, gender, position and event. The data for athletes consists of a general data part and an event specific data part. The archery athlete data includes general data, handedness, MQS (Mininum Qualiciation Score), world ranking, individual event, and team event. The basketball athlete data includes general part, IFID, uniform number, uniform color, position, and type of qualification. All of the data will be checked by using Java servlets when the associated user interface page is submitted to the server.

3.2 Database Schema Conceptual Model - Entity-Relationship Diagram

According to data analysis above, the schema for the ODRS database contains the following entities: users, officials, athletes, archery, basketball and idstore. All the entities and attributes are detailed in Figure 2.

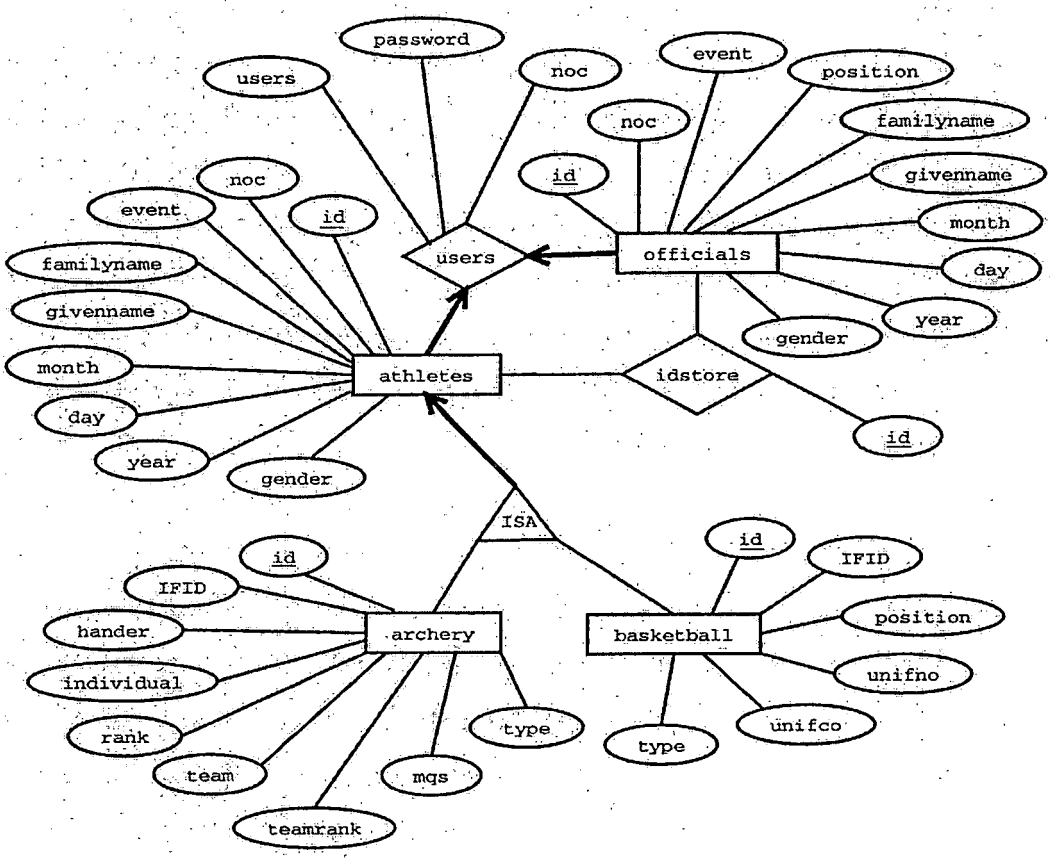

Figure 2. Entity-Relationship Diagram

3.3 Database Schema Logical Model - Relational Schema

The conceptual model ER diagram maps into the following relational table design. In the following tables, underlined fields indicate the primary key.

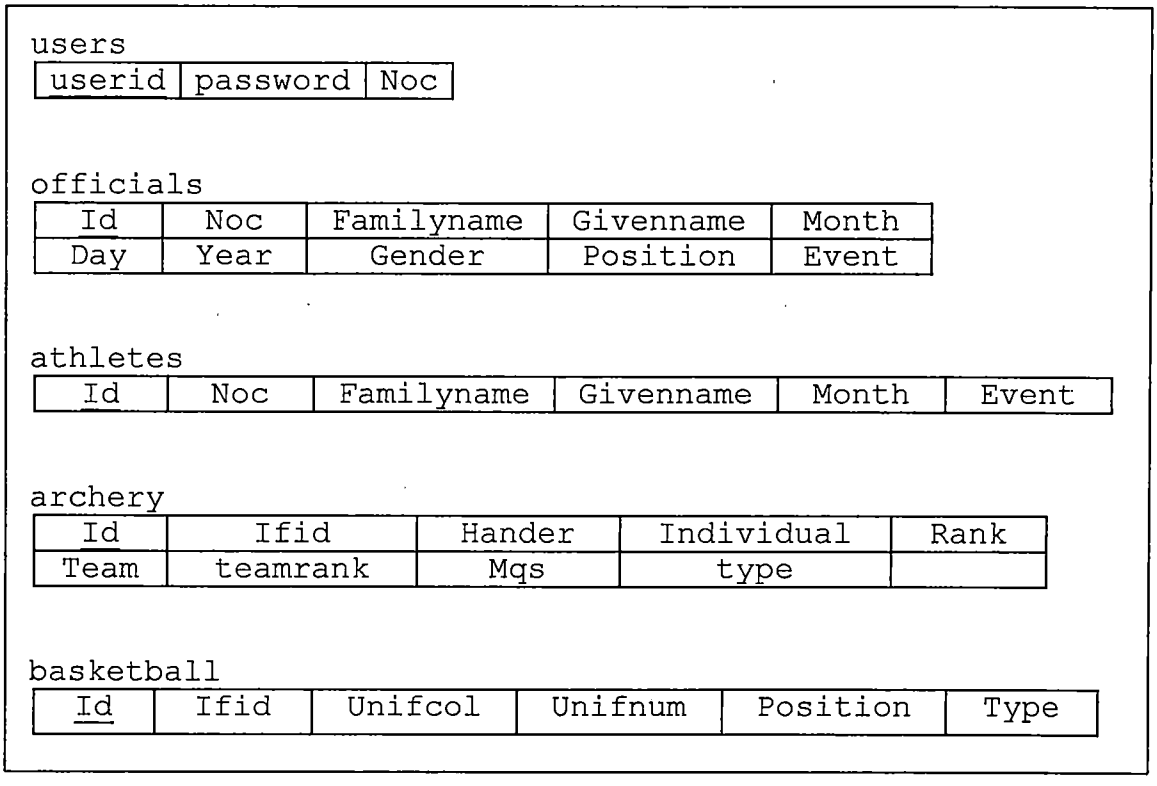

Figure 3. Olympiad Delegation Registration System Database Relational Schema

3.4 Data Type and Details

The logical model establishes the following detailed design in PostgreSQL database. The following tables describe data type, length, primary key, null or non-null keys, and extra information, such as auto_increment.

Table 1. Structure of Table Users

Field	Type	Null	key	Default	Extra
userid	varchar(15)		PRI		
password	varchar(15)				
noc	varchar(50)				

Table 2. Structure of Table Officials

Field	Type	Null	Key	Default	Extra
id	char(9)		PRI		
noc	char(3)				
familyname	varchar(25)				
givenname	varchar(25)				
month	varchar(2)				
dday	varchar(2)				
year	varchar(4)				
gender	varchar(1)				
position	varchar(25)				
event	varchar(25)				

Table 3. Structure of Table Athletes

Field	Type	Null	Key	Default	Extra
id	char(9)		PRI		
noc	char(3)				
event	char(2)				
familyname	varchar(25)				
givenname	varchar(25)				
month	varchar(2)				
day	varchar(2)				
year	varchar(4)				
gender	varchar(1)				

Table 4. Structure of Table Archery

Field	Type	Null	Key	Default	Extra
Id	char(9)		PRI		
Ifid	varchar(16)	Yes			
Rank	char(3)				
Individual	char(4)				
Team	char(4)	Yes			
Teamrank	char(3)	Yes			

hander	char(1)				
Mqs	varchar(3)				
Type	varchar(3)				

Table 5. Structure of Table Basketball

Field	Type	Null	Key	Default	Extra
Id	char(9)		PRI		
Ifid	Varchar(16)	Yes			
Unifnum	varchar(2)				
Unifcol	Varchar(10)			1	
Position	Varchar(10)				
Type	Varchar(3)				

Table 6. Structure of Table Idstore

Field	Type	Null	Key	Default	Extra
Id	char(9)		PRI		

CHAPTER FOUR

PROJECT IMPLEMENTATION

ODRS is designed to perform nine different functions for two different users. Figure 4 is the Use Case Diagram of this project.

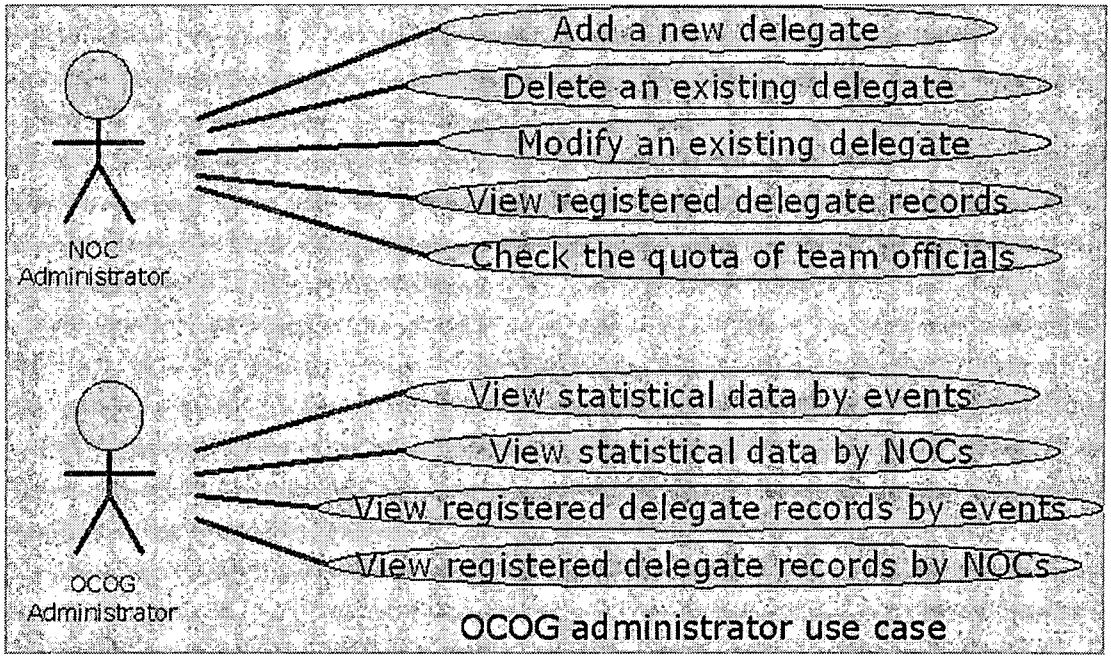

Figure 4. User Case Diagram

4.1 Olympiad Delegation Registration System Graphical User Interface Design

ODRS GUI is easy to use. The GUI is written using Hyper Text Markup Language (HTML). All the functions that the user has are placed in the menu part, which is the upper region

of the page. ODRS uses JavaServelets to check the user input accuracy. All the inputs that are not acceptable by the system will be reported as errors with the getAttribute function. The following sub-sections explain the details of how the GUI works.

4.1.1 Olympiad Delegation Registration System Home Page

This page will be the ODRS first page. All of visitors no matter who they are can view the page. However, only ODRS users can access the ODRS with the ODRS users filter. According to NOC users or OCOG users, after the user clicks on related to hyper link (NOCs users or OCOG user), ODRS forwords to the login page.

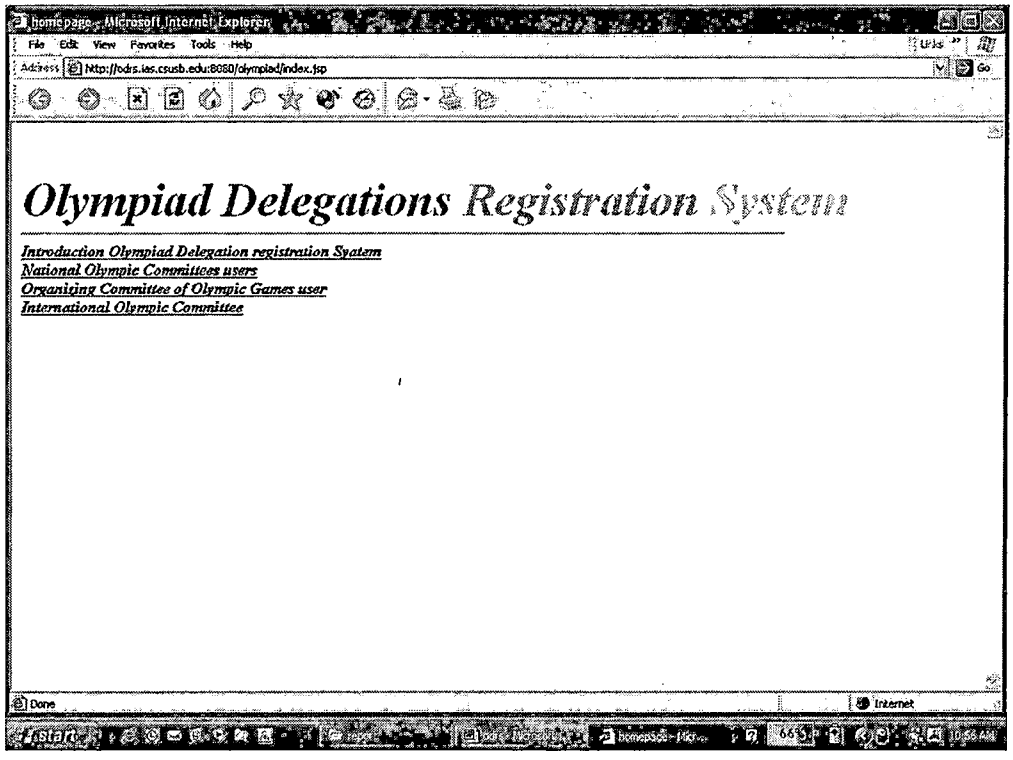

Figure 5. Olympiad Relegation Registration System Home Page

4.1.2 Olympiad Delegation Registration System Users Login Page

The user logs in by providing a user id and a password that are created previously by the administrator. After the servlet verifies the user id and password, it forwards to an index page according to whether the user is an NOC user or an OCOG user. Moreover, the user information will be saved in the session for later use, and the session will be killed when the browser is closed or when the user is idle for 1800 seconds (30 minutes), which is the default session lifetime set up by Tomcat Server. The system also allows the users

to change their user id and password after the first login success.

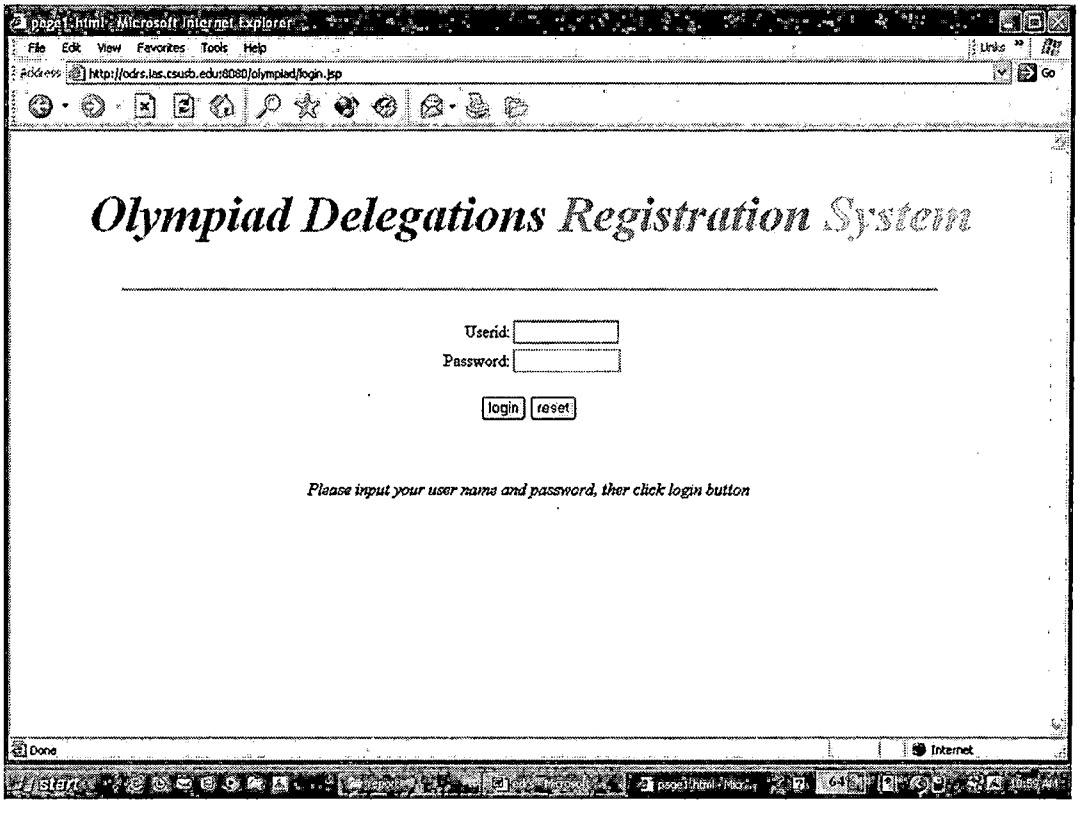

Figure 6. Olympiad Delegation Registration System User Login Page

4.1.3 National Olympic Committees Index Page

After the NOC user login success, ODRS forwards the NOC user to an NOC index page, which is responsable to its own delegation. In that page, the user can chose add, modify, or delete a delegate function, or the user can view the delegation list ordered by name or by events. In that page,

the user also can click the change password and userid link to change these.

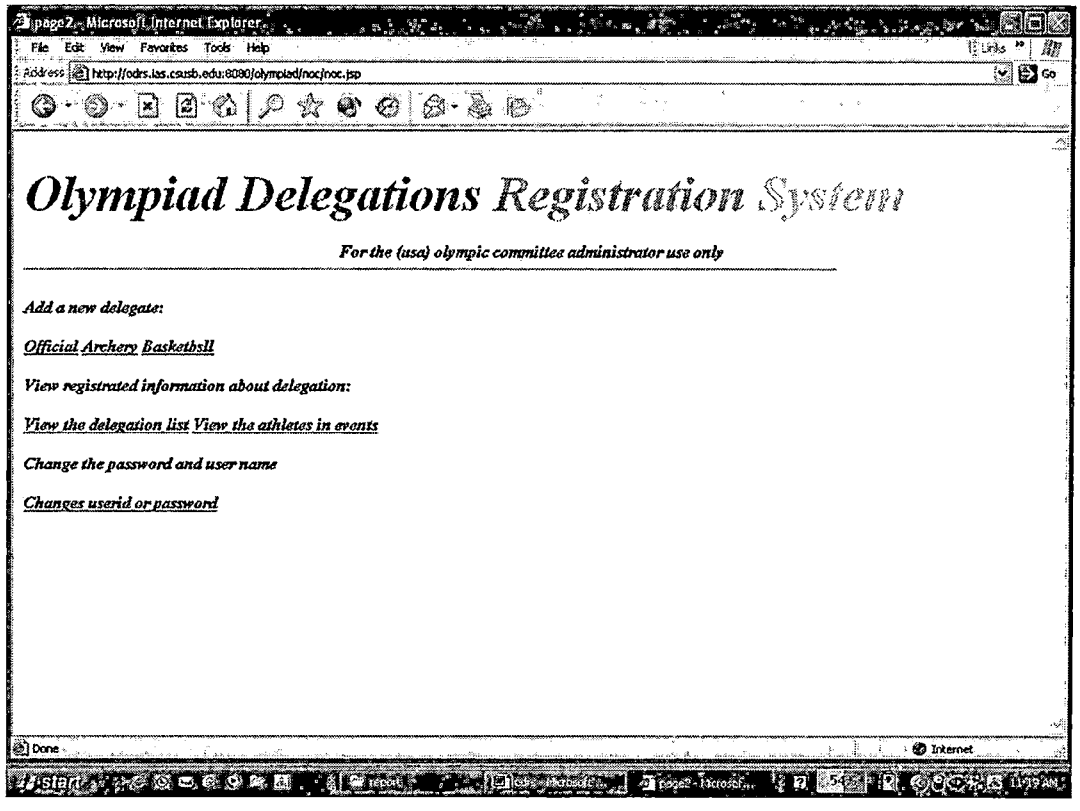

Figure 7. National Olympic Committees User Page

4.1.4 Organizing Committee of the Olympic Games Index Page

After the OCOG user login success, ODRS forwards OCOG users to an OCOG index page in which the OCOG user can view the statistical data about the delegation registrations of all the NOCs. In that page, the user can chose different formatting layouts according to individual event or NOC.

Also in that page, the user can click on the change password and userid link to change these as they like.

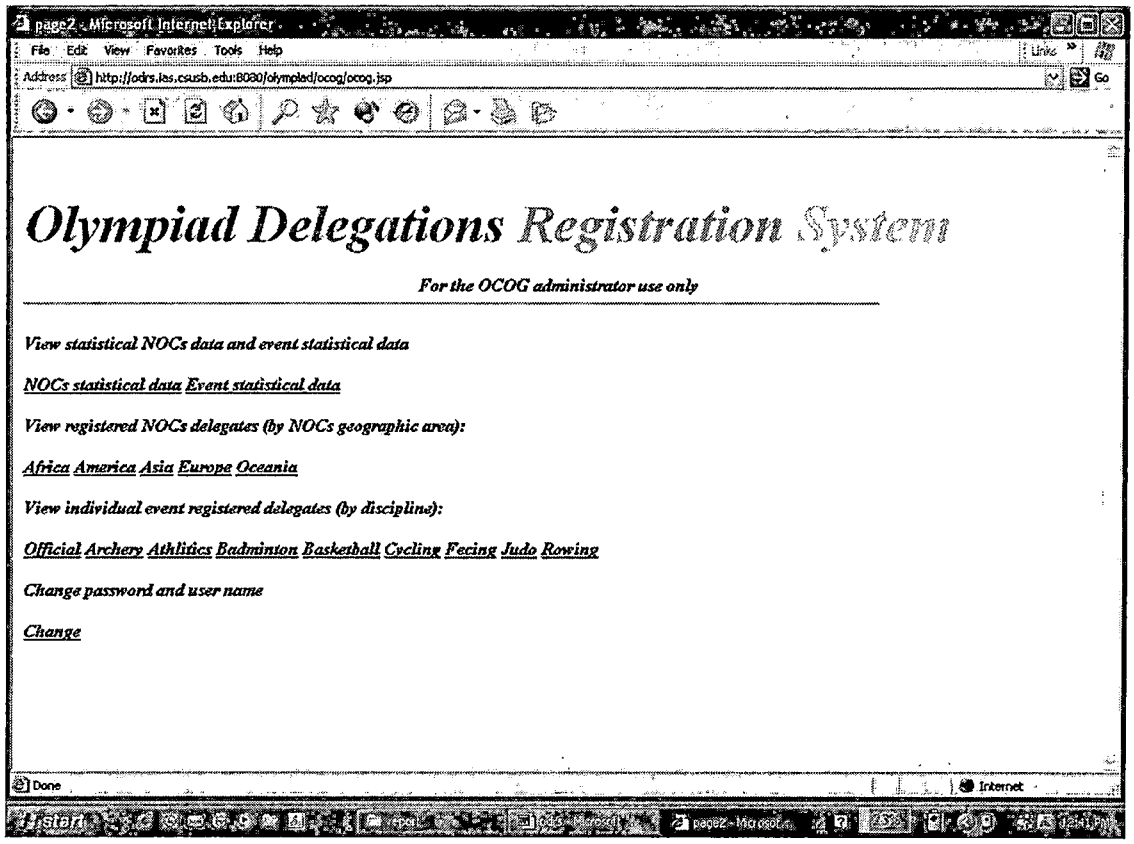

Figure 8. Organizing Committee of the Olympic Games index page

4.1.5 View Delegate List Page (by Delegation)

When the NOC user clicks the view delegation list in the NOC index page, the user is brought to its own delegation list page. In this page each delegate is automaticaly assigned a unique ID number. When the user

clicks the Id representing an individual delegate, the system forwards to an individual information view page.

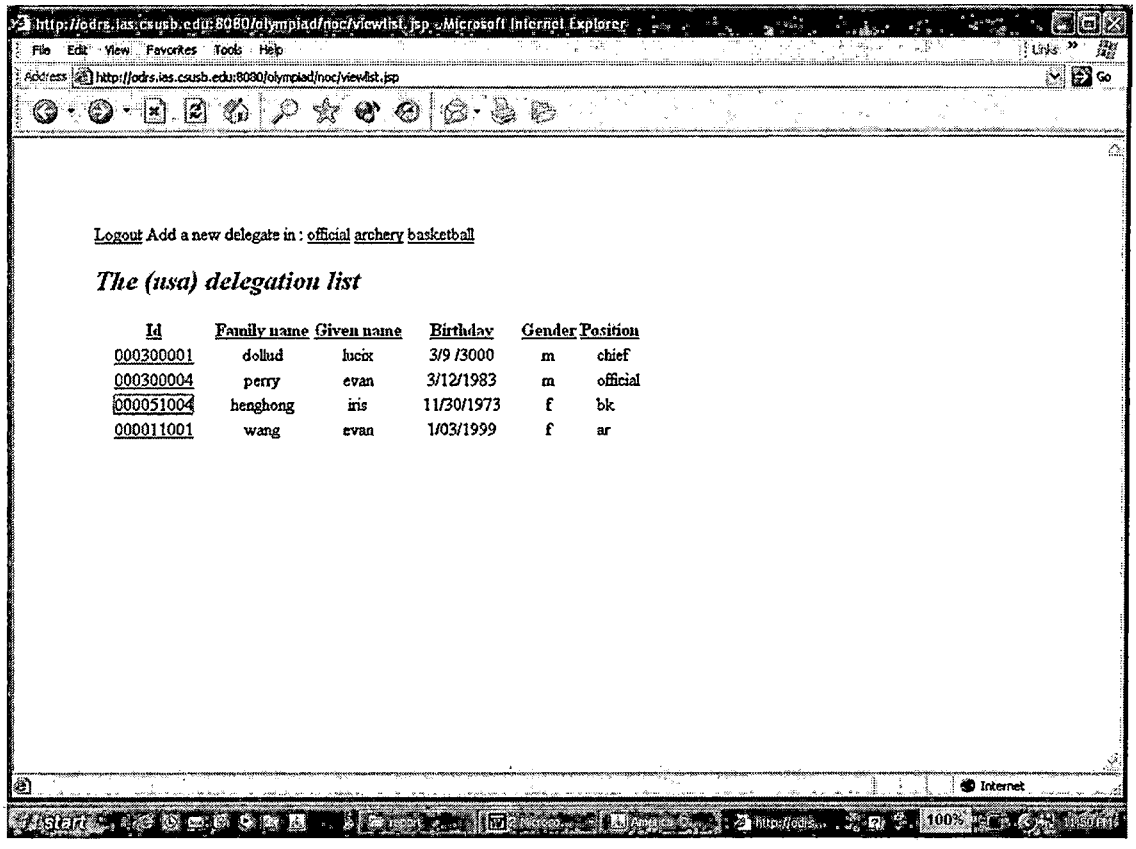

Figure 9. View Delegate List Page (by Delegation)

4.1.6 View Delegate List (by Event)

According to Olympiad convention, each event should have a list to show subject event delegates registered. Thus, this page includes sub-events, delegate ids, family names, given names and genders.

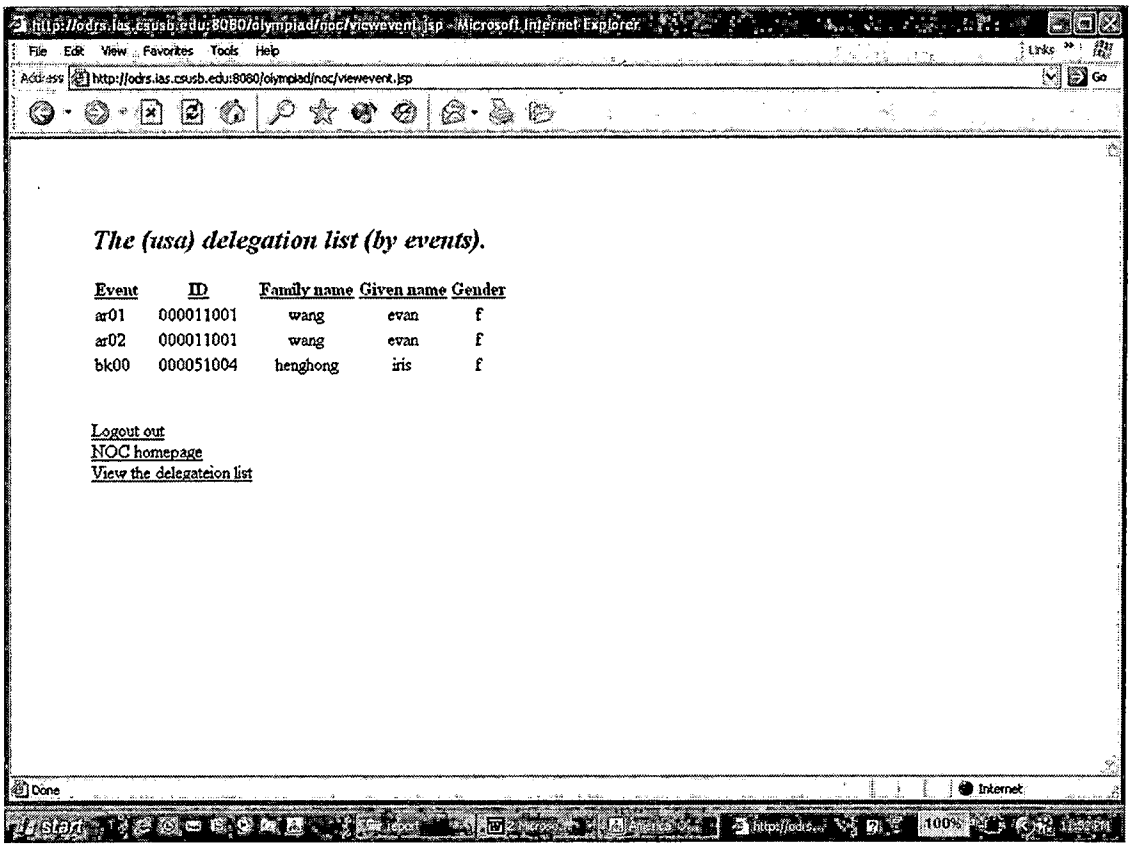

Figure 10. View the Delegate List Page (by Event)

4.1.7 Add Archery Delegate Page

This page is specifically for adding delegates who are athelets participating in the archery competition. To add a delegate in archery, the NOC user inputs the delegate's family name, given name, day of birth, gender, IFID, handedness, minimum qualification score, team participant, and world ranking. Some of above data is selected from lists, so that the user just needs to choose the select item they want. Each of the forms uses a servlet to check error. If

there is an error, the servlet can prompt a message with the getAttribute function.

4.1.8 Add Basketball Delegate Page

This is the other add event delegate page. The input names include family name, given naam, day of birth, gender, IFID, uniform color, uniform number, position, and type of qualification. ODRS uses a servlet to check the input error. If there is an error during input, the system automically reports the error and prompts the user for the correct value.

4.1.9 Add Official Delegate Page

This page is to add official delegates in each Olympic delegation. Official categories includes: chefs de mission, deputy chef de mission, team officials and coach. In accordance with the IOC's Team Official quotas, allowed total number of team officials depends on the total of the number of athletes.

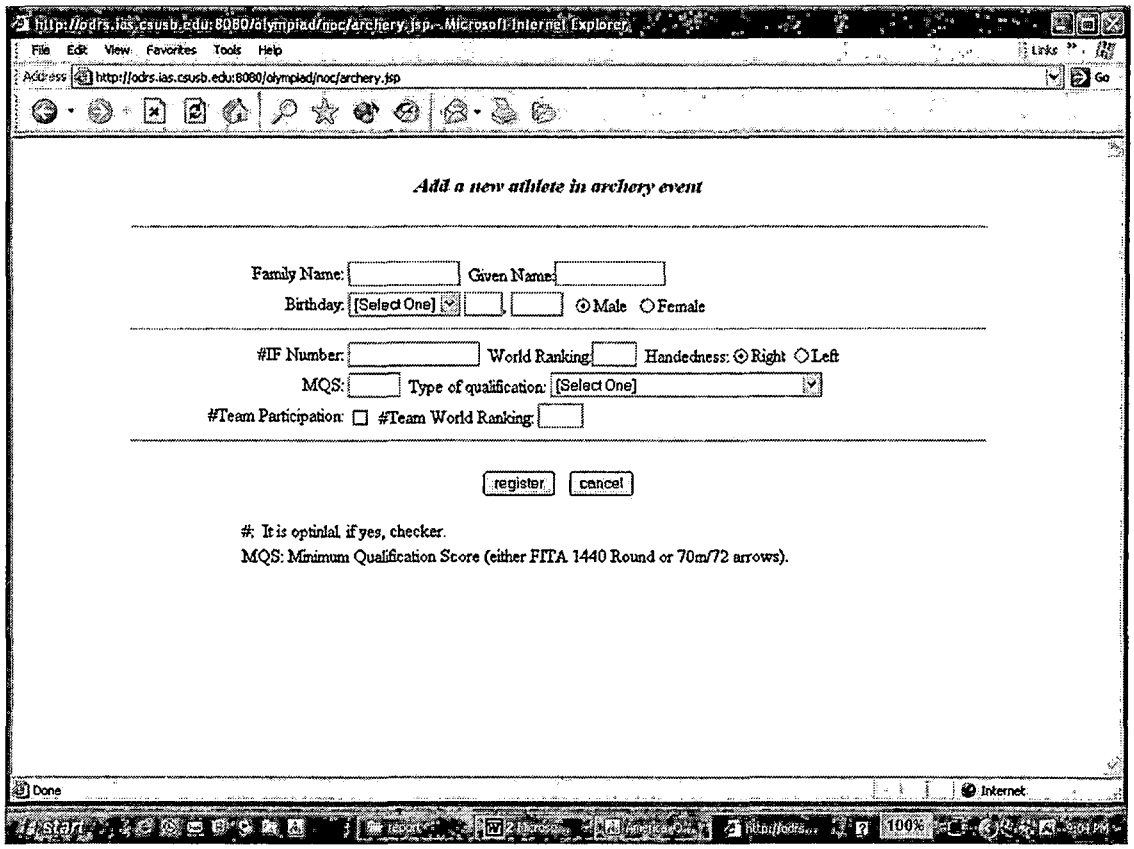

Figure 11. Add Archery Delegate Page

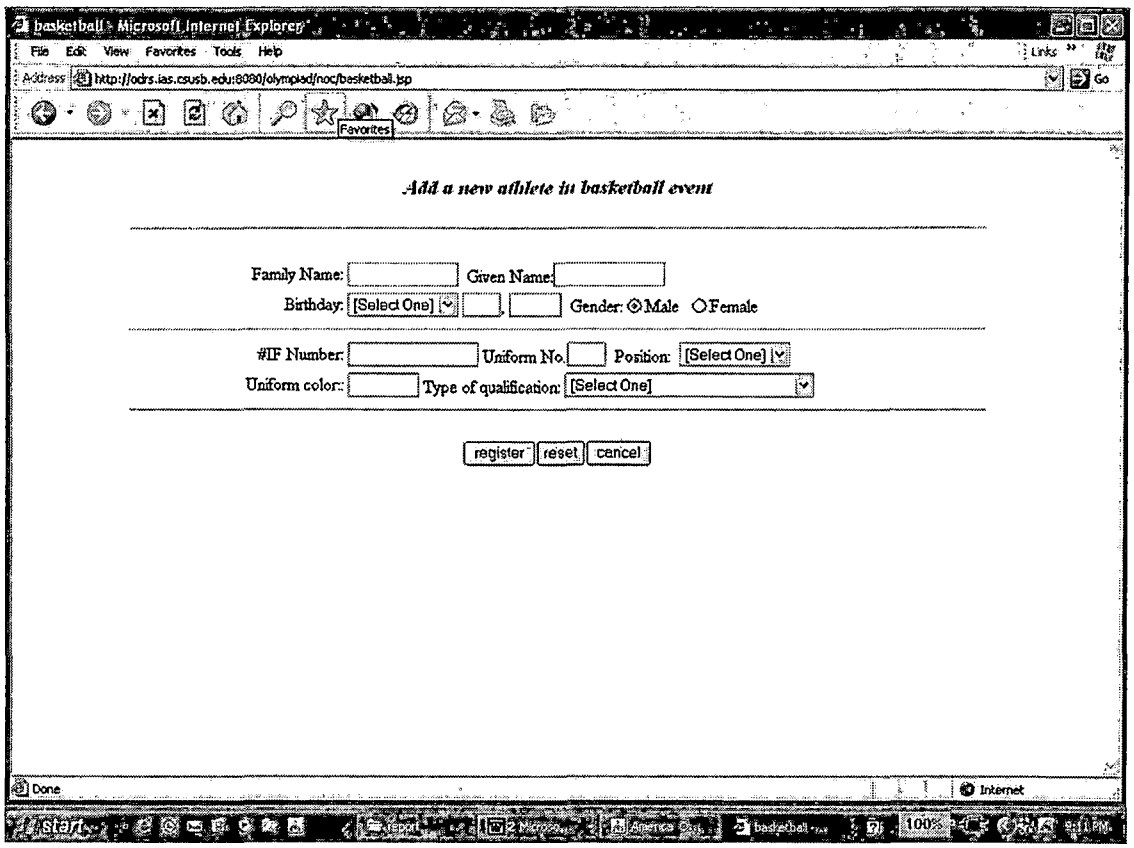

Figure 12. Add Basketball Delegate Page

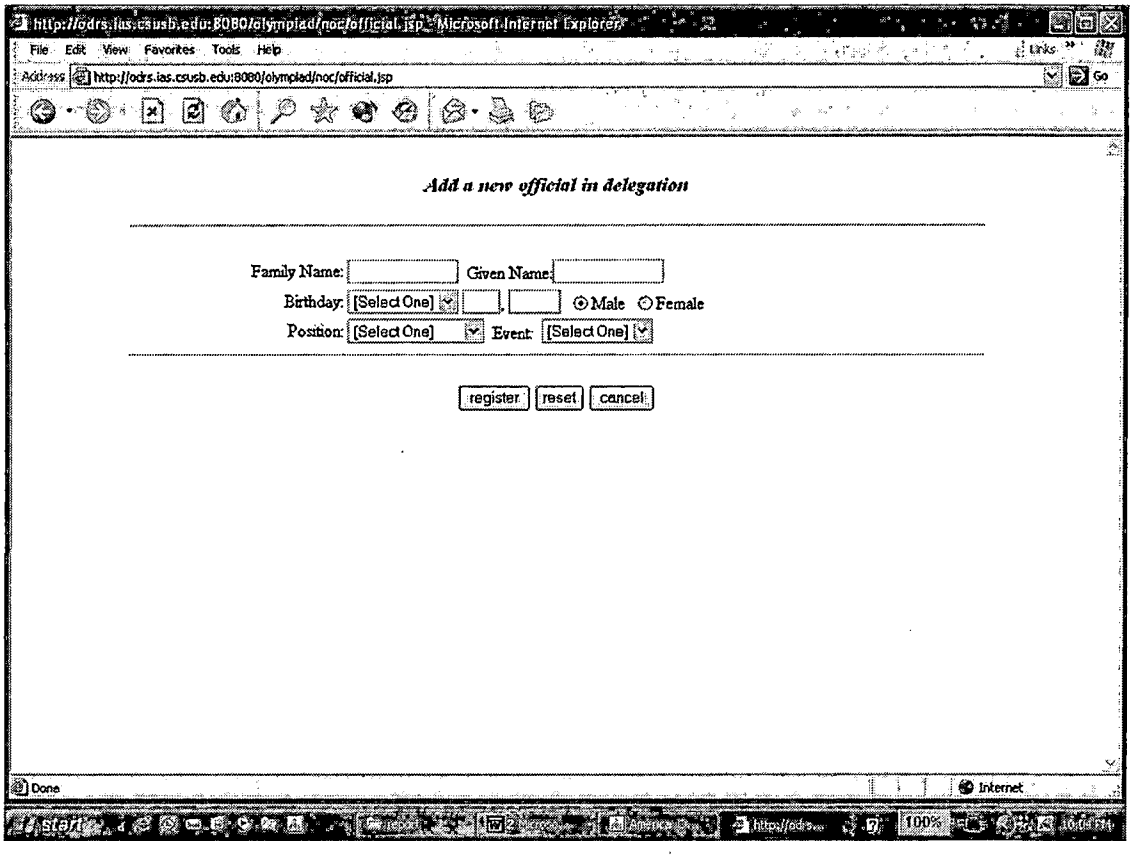

Figure 13. Add Official Delegate Page

4.1.10 View the Individual Delegate Page

Once the user clicks the view list page, ODRS forwards a detailed individual information page according to the category selected: official, archery, or basketball delegate. In each view page there are three submit buttons to delete, modify and cancel. If the user selects the delete button, then the user is sent to a confirmation page.

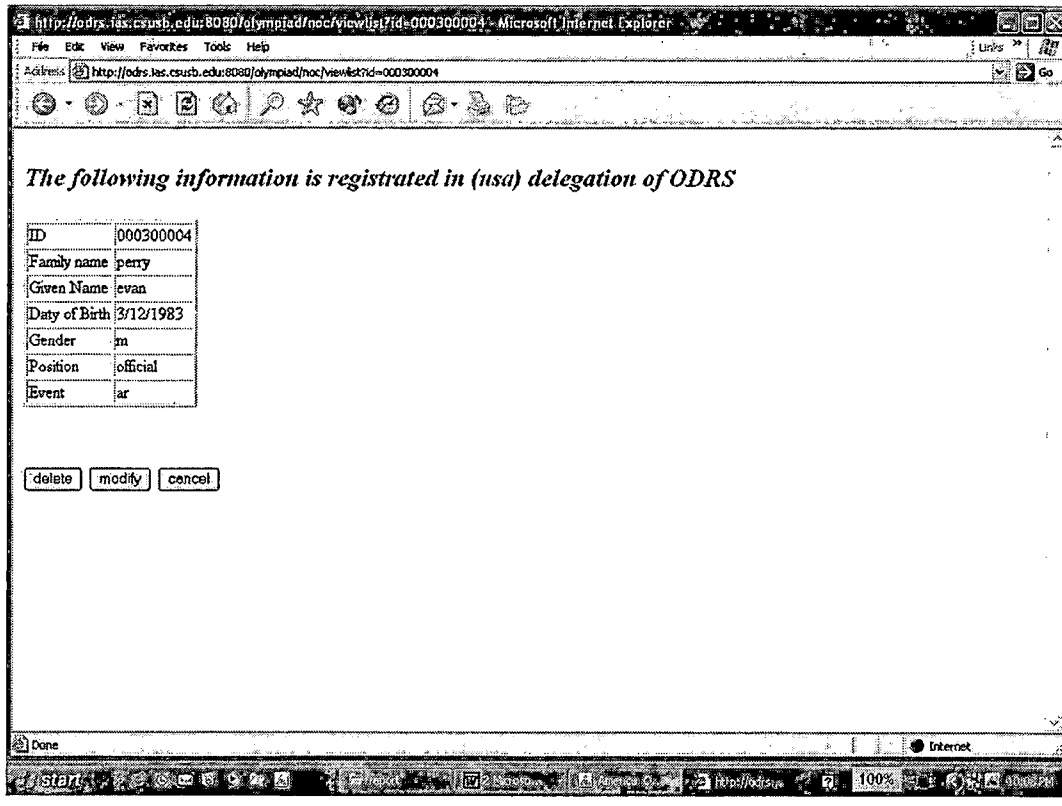

Figure 14. View the Individual Delegate Page

4.1.11 Modify Individual Delegate Page

In this page, once NOC user clicks on modify button, the use is brought to modify deletate page related to special event. This page is simile to add delegate page, only all of the delegate information is forwarded. After the use changes some items and then click on modify button. The the special delegate information was changed.

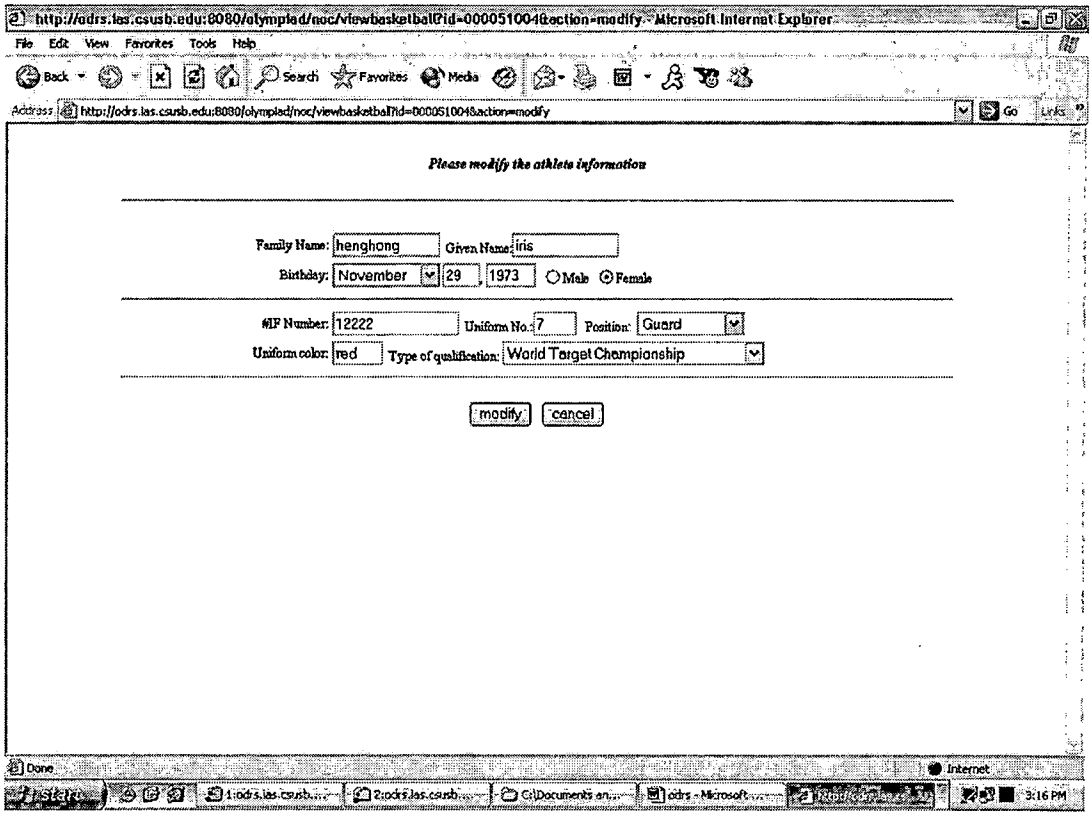

Figure 15. Modify Individual Delegate Page

4.1.12 Delete Individual Delegate Page

This is delete individual delegate page. After NOC user clicks delete button of View Individual Delegate Page, the user is brought to a delete confirm page. Once the user click on the delete button again on this page, all of the special delegate information will be deleted from ODRS database and the user is forwarded to a View Delegation list page.

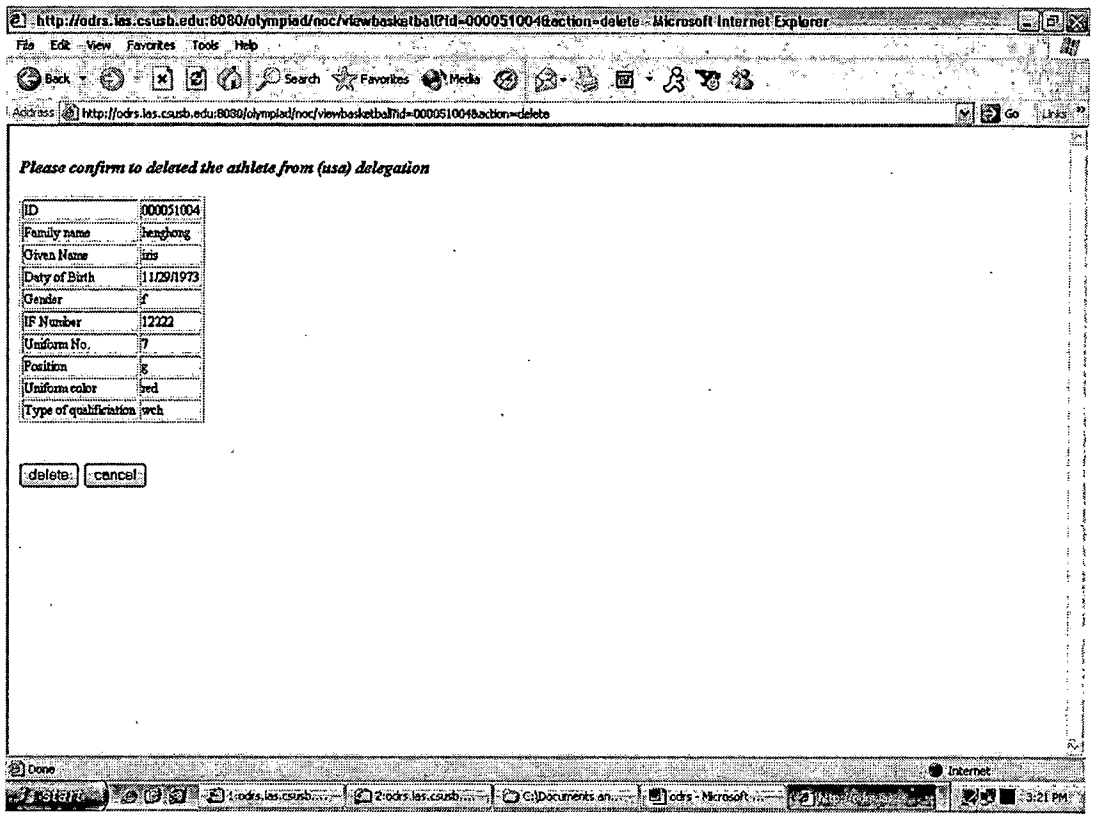

Figure 16. Delete individual Delegate Page

CHAPTER FIVE

SYSTEM VALIDATION

The system validation test is a kind of test process, which can ensure that our program meets the expectation of the user. The purpose of the system validation is to provide a high degree of assurance that a specific process will consistently produce a result which meets predetermined specifications and quality attributes. This can also guarantee the system performance and reliability.

5.1 Unit Testing

Unit test is the basic level of testing where individual components are tested to ensure that they operate correctly. These individual components can be object, class, program, etc. The unit testing results of WAOCC are shown in Table 4.

Table 7. Unit Test Results (Forms)

Forms	Tests Performed	Results
ODRS home page	• Check all links are effect properly.	Pass
Login Page	• Check all the button work	Pass

	properly. • Verify the page can get the error message and work properly by the message. • Verify the user save in session after login	
Logout Page	• Check all the button work properly. • Verify the user remove from session after logout. • Check the page redirect to proper page after logout.	Pass
Chang password and userid page	• Verify handling input data is correct. • Makd sure relative data are changed	Pass
NOCs index Page	• Verify after login the page showing the correct user session information. • Check each links work properly.	Pass

View official information page	• Verify all information showing correctly. • Check all buttons work properly.	Pass
View archery information page	• Verify all information showing correctly. • Check all buttons work properly.	Pass
View basketball information page	• Verify all information showing correctly. • Check all buttons work properly.	Pass
Add official page	• Verify handling valid datd input. • Check all the buttons work properly.	Pass
Add archery page	• Verify handling valid data input • Check all the buttons work properly.	Pass
Add basketball page	• Verify handling valid data input.	Pass

	• Check all the buttons work properly.	
Moidify official Page	• Verify retrieval data are correct. • Check all the buttons work properly. • Check all the modified data are updated.	Pass
Modify archery page	• Verify retrieval data are correct. • Check all the buttons work properly. • Check all the modified data are updated.	Pass
Modify basketball page	• Verify retrieval data are correct. • Check all the buttons work properly. • Check all the modified data are updated.	Pass
Delete official page	• Verify all the data are deleted.	Pass

	• Check all the buttons work properly.	
Delete archery page	• Verify all the data are deleted. • Check all the buttons work properly.	Pass
Delete basketball page	• Verify all the data are deleted • Check all the buttons work properly.	Pass
View delegation list Page	• Verify the users except own NOC delegation user can not view this page. • Verify all the user relative information is correct. • Verify all the user relative information shown in the list. • Make sure the ID link of individual delegate works properly.	Pass

View event lsit Page	• Verify the context in the view event list page is correct. • Check all the buttons works properly.	Pass

Table 8. Unit Test Results (Class: Official)

Functions	Tests Performed	Results
insertDelegate	• Check if the data can insert database correct.	Pass
findDelegate	• Verify it gets all information from database correctly. • Make sure it return data is correct.	Pass
deleteDelegate	• Verify it deletes all the delegate information.	Pass
updateDelegate	• Check it can modify data about the delegate properly.	Pass

Table 9. Unit Test Results (Class: OfficialServlet)

Forms	Tests Performed	Results
Init	• Make sure if no page exists, it throws correct message.	Pass
Dopost	• Verify all the attribute is gotten properly and correctly. • Make sure all the containers are handled. • Verify the session of the login user is handled properly. • Make sure set all attributes is proper. • Check getIdNumber return correct value. • Verify the forward page is correct	Pass

Table 10. Unit Test Results (Class: ViewOfficialServlet)

Functions	Tests Performed	Results
Init	• Make sure it throws message is correct.	Pass
Doget	• Make sure get all parameters are correct. • Make it set the session attribute is correct. • Verify find delegate date is correct. • Check it forward all pages is properly.	Pass

Table 11. Unit Test Results(Class:ModifyOfficialServlet)

Functions	Tests Performed	Results
Init	• Make sure it throws message is correct.	Pass
Dopost	• Make sure it can get all parameters properly.	Pass

Functions	Tests Performed	Results
	• Verify modifying data is corrct. • Verify find delegate date is correct. • Check it forward all pages are properly.	

Table 12. Unit Test Results(Class:DeleteOfficialServlet)

Functions	Tests Performed	Results
Init	• Make sure it throws message is correct.	Pass
Doget	• Make sure get correct id to delete. • Make it gets all parameters properly. • Check it forward page is proper.	Pass

Table 13. Unit Test Results (Class: Archery)

Functions	Tests Performed	Results
insertDelete	• Check if the data can insert database correct.	Pass
findDelegate	• Verify it gets all information from database correctly. • Make sure it return data is correct.	Pass
delegateDelegate	• Verify it deletes all the delegate information.	Pass
updateDelegate	• Check it can modify data about the delegate properly.	Pass

Table 14. Unit Test Results (Class: ArcheryServlet)

Forms	Tests Performed	Results
Init	• Make sure if no page exists, it throws correct message.	Pass

Forms	Tests Performed	Results
Dopost	• Verify all the attribute is gotten properly and correctly. • Make sure all the containers are handled. • Verify the session of the login user is handled properly. • Make sure set all attributes is proper. • Check getIdNumber return correct value. • Verify the forward page is correct	Pass

Table 15. Unit Test Results (Class: ViewArcheryServlet)

Functions	Tests Performed	Results
Init	• Make sure it throws message is correct.	Pass

Functions	Tests Performed	Results
Doget	• Make sure get all parameters are correct. • Make it set the session attribute is correct. • Verify find delegate date is correct. • Check it forward all pages are properly.	Pass

Table 16. Unit Test Results (Class:ModifyArcheryServlet)

Functions	Tests Performed	Results
Init	• Make sure it throws message is correct.	Pass
Dopost	• Make sure it can get all parameters properly. • Verify modifying data is corrct. • Verify find delegate date	Pass

Functions	Tests Performed	Results
	is correct. • Check it forward all pages are properly.	

Table 17. Unit Test Results (Class:DeleteArcheryServlet)

Functions	Tests Performed	Results
Init	• Make sure it throws message is correct.	Pass
Doget	• Make sure get correct id to delete. • Make it gets all parameters properly. • Check it forward page is proper.	Pass

Table 18. Unit Test Results (Class: Basketball)

Functions	Tests Performed	Results
insertDelete	• Check if the data can insert database correct.	Pass
findDelegate	• Verify it gets all information from database correctly. • Make sure it return data is correct.	Pass
delegateDelegate	• Verify it deletes all the delegate information.	Pass
updateDelegate	• Check it can modify data about the delegate properly.	Pass

Table 19. Unit Test Results (Class: BasketballServlet)

Forms	Tests Performed	Results
Init	• Make sure if no page exists, it throws correct message.	Pass

Forms	Tests Performed	Results
Dopost	• Verify all the attribute is gotten properly and correctly. • Make sure all the containers are handled. • Verify the session of the login user is handled properly. • Make sure set all attributes is proper. • Check getIdNumber return correct value. • Verify the forward page is correct	Pass

Table 20. Unit Test Results(Class:ViewBasketballServlet)

Functions	Tests Performed	Results
Init	• Make sure it throws message is correct.	Pass

Functions	Tests Performed	Results
Doget	• Make sure get all parameters are correct. • Make it set the session attribute is correct. • Verify find delegate date is correct. • Check it forward all pages is properly.	Pass

Table 21. Unit Test Results (Class: ModifyBasketballServlet)

Functions	Tests Performed	Results
Init	• Make sure it throws message is correct.	Pass
Dopost	• Make sure it can get all parameters properly. • Verify modifying data is corrct. • Verify find delegate date	Pass

Functions	Tests Performed	Results
	is correct. • Check it forward all pages are properly.	

Table 22. Unit Test Results
(Class:DeleteBasketballServlet)

Functions	Tests Performed	Results
Init	• Make sure it throws message is correct.	Pass
Doget	• Make sure get correct id to delete. • Make it gets all parameters properly. • Check it forward page is proper.	Pass

Table 23. Unit Test Results (Class: User)

Forms	Tests Performed	Results
Find	• Make sure the returned type and data are correct.	Pass
Update	• Make it can access database and updated datasure are correct.	Pass

Table 24. Unit Test Results (Class: ChangePasswordServlet)

Forms	Tests Performed	Results
Init	• Make sure it can throws message is proper.	Pass
Dopost	• Make sure it can get all parameters properly. • Verify finded userid and password are correct. • Check it forwards the pages is correct.	Pass

Table 25. Unit Test Results (Class: Constants)

Forms	Tests Performed	Results
Constants	• Make sure all static final strings are initialized properly.	Pass

Table 26. Unit Test Results (Class: IdNumber)

Forms	Tests Performed	Results
getIdNumber	• Make it can calcult correct id number. • Make returned type and data is properly.	Pass

Table 27. Unit Test Results (Class: EventId)

Forms	Tests Performed	Results
EventId	• Make sure all static final strings are	Pass

Forms	Tests Performed	Results
	initialized properly.	

Table 28. Unit Test Results (Class: LoginServlet)

Forms	Tests Performed	Results
Init	• Make sure it can throws message properly.	Pass
Dopost	• Make sure it can get all parameters properly. • Verify it can find correct userid and password and return properly. • Verify the user save in session after login. • Check it can forward the relative pages is correct.	Pass

Table 29. Unit Test Results (Class: LogoutServlet)

Forms	Tests Performed	Results
Init	• Make default is proper.	Pass
Doget	• Make sure set all session invalidate. • Make suer sedredirect page correct.	Pass

Table 30. Unit Test Results (Class: NocUserFilter)

Forms	Tests Performed	Results
Init	• Make default is proper.	Pass
Destroy	• Make sure default correctly.	
Dofilter	• Make sure it get user attribute correctly. • Verify sedredirect page correctly.	Pass

Table 31. Unit Test Results (Class: ViewList)

Forms	Tests Performed	Results
ViewList	• Make initial constructor to be proper.	Pass
FindDelegate	• Make sure it access databae correctly. • Verify it get data correctly. • Make sure returned type is correct.	Pass

Table 32. Unit Test Results (Class: ViewListServlet)

Forms	Tests Performed	Results
Init	• Make sure it can throws message properly.	Pass
Doget	• Make sure it can get all parameters properly. • Verify it find delegate correctly.	Pass

Forms	Tests Performed	Results
	• Verify it set session with deficient event correctly. • Check it can forward the relative pages properly.	

5.2 Subsystem Testing

Subsystem testing is the next step in the testing process, where all related units from a subsystem do a certain task. Thus, the subsystem test process is useful for detecting interface errors and specific functions. Table 24 shows subsystem test results in detail.

Table 33. Subsystem Test Results

Subsystem	Tests Performed	Results
Authorize subsystem	• Test if subsystem can get the error message. • Make sure the result of authorizing user is correct. • Verify the login user information is store in session properly.	Pass

Subsystem	Tests Performed	Results
	• Check if the saving user login information function stores the user information in cookies for future login use. • Verify the login page redirect to the correct browsing request after the user logins in. • Check if users can update their own account properly.	
data management subsystem	• Make sure the subsystem checks the user before dealing with add, modify, or delete function. • Verify the subsystem check the user privilege before handling the deletgate's data. • Verify if the subsystem shows the page related to view delegate data is the	Pass

Subsystem	Tests Performed	Results
	users are the owner.	
Qualification subsystem	• Verify subsystem assigns each delegate with unique Id number. • Make sure if subsystem can check each NOCs user qualify the rule of Olympaid Registration. • Verify each delegate qualify the special event registeation rule.	Pass
Browsing subsystem	• Check if the subsystem checks for user privilege before showing pages. • Verify the page is showing properly after the user click on the page link.	Pass

5.3 System Testing

System testing is the testing process that uses real data to test the system. The data are what the system is intended to manipulate. The ODRS system testing steps

includes three steps: first ingegrating all subsystem into ODRS; second starting up all the server which ODRS need to run such Tomcat server and PostgreSQL database server, finally, by using a variety of data run the ODRS to analyze the overall result. ODRS system testing steps as following:

Table 34. System Test Results

System Testing	Results
1. Install ODRS system into web server.	Pass
2. Start up Tomcat server, PostgreSQL database server.	Pass
3. Running testing by using real data on all forms and reports.	Pass

CHAPTER SIX

MAINTENANCE MANUAL

In order that the system works smoothly and meets the expectation of the users, it is important to make a maintenance manual with a system. The maintenance manual records any information that can be used to setup the system or backup the system. ODRS maintenance manual includes Software installation, variable installation and ODRS installation three major issuses

6.1 Software Installation

ODRS requires RedHat Linux 9, JSDK (Java 2 Platform, Standard Edition kit), Ant, Tomcat, PostgreSQL and JDBC to run the programs. The detailed installation of these softwares explains explicitly as following.

6.1.1 RedHat Installation

The following are the approximate sequence of steps to install Redhat Linux 9.0 for use as a server platform for Java-based web applications. We assume installation of Linux 9.0 with CD-ROM.

1. The hardware compatibility and disk space are particularly important before installation. Make sure a

server installation require 850MB for a minimal installation without X Windows and unnecessary package groups.

2. Redhat Linux 9.0 is distributed on 3 CDs. Start by booting the system with the first CD in the drive, making sure the bios is set to boot from CD before other devices.

3. Follow the install wizard and sets up the required information such as network setting and the hardware environment.

4. After configure and select indivisual the necessary packages, then install it, the machine will restart and Redhat is installed.

6.1.2 Inatall JAVA (J2SE)

J2SE is the compiler program for JSP and JAVA Servlet programs. It is required in Tomcat JAVA Container. To install JAVA, first thing we need to create a directory under which you will install java systems. We use following commands to create a directory named java under "/sur" directory. The following (6.1.3 Install Ant and 6.1.4 Install Tomcat) software installation are default that directory.

```
mkdir /usr/java/
cd /usr/java/
```

To retrieve a copy of the of java2 SDK RPM, normally we can go to http://java.sun.com to download SDK. Then we execute the following commands:

```
chmod +x j2sdk-1_4_2_03-linux-i586-rpm.bin
./j2sdk-1_4_2_03-linux-i586-rpm.bin
rpm -ivh j2sdk-1_4_2_03-linux-i586.rpm
```

And, set the environment variables JAVA_HOME and modify the PATH variable in the file /etc/profile.d/myenv.sh:

```
JAVA_HOME=/usr/java/j2sdk1.4.2_03
PATH=${PATH}:${JAVA_HOME}/bin
Export JAVA_HOME
```

6.1.3 Install Ant

Apache Ant is a Java-based build tool. It is similar to make. Ant is extended using Java classes. Instead of writing shell commands, the configuration files are XML-based, calling out a target tree where various tasks get executed.

To retrive a compressed achive of ant, normally go to http://ant.apache.org and obtain a more recent URL with tar.gz file. Then we execute following commands to uncompress the achive files.

```
tar -zxvf apache-ant-1.6.0-bin.tar.gz
```

And, set the environmental variable ANT_HOME and modify /etc/profile.d/myenv.sh to have the following contents:

```
JAVA_HOME=/usr/java/j2sdk1.4.2_03
```

```
ANT_HOME/usr/java/apache-ant-1.6.0
PATH=${PATH}:${JAVA_HOME}/bin/:${ANT_HOME}/bin
export JAVA_HOME
export ANT_HOME
export PATH
```

6.1.4 Install Tomcat

Tomcat is one of the apache jakarta projects, which is a web container to process JSP and JAVA Servlet programs and to serve static web pages.

To retrive a compressed achive of Tomact and extract, normally we go to http://jakarta.apache.org/tomcat, and obtain a more recent URL. Then we execute following commands to uncompress the achive files.

```
tar -xzvf jarkata-tomcat-4.1.29.tar.gz
```

And also, set the environment variable by adding the following lines in the file /etc/profile.d/myenv.sh

```
JAVA_HOME=/usr/java/j2sdk1.4.2_03
ANT_HOME/usr/java/apache-ant-1.6.0
CATALINA_HOME=/usr/java/jarkata-tomcat-4.1.29
PATH=${PATH}:${JAVA_HOME}/bin/:${ANT_HOME}/bin:
${CATALINA_HOME}/bin
export JAVA_HOME
export ANT_HOME
export CATALINA_HOME
```

```
export PATH
```

After we install Tomcat and set the environmental variable, we need to go to Tomcat's configuration directory and substitute server.xml file with following contents:

```
<Server port="8005" shutdown="SHUTDOWN" debug="0">
    <Service name="Tomcat-Standalone">
    <Connector
className="org.apache.catalina.connector.http.HttpConnector"
    port="8080"
    minProcessors="5"
    maxProcessors="75"
    enableLookups="false"
    redirectPort="8443"
    acceptCount="10"
    debug="0"
    connectionTimeout="60000" />
    <Engine name="Standalone" defaultHost="localhost"
debug="0">
    <Host name="localhost"
        debug="0"
        appBase="webapps"
        unpackWARs="true"
        autoDeploy="true"
        deployXML="true"
```

```
            liveDeploy="true">
          <Context path="" docBase="ROOT" debug="0" />
          </Host>
          </Engine>
        </Service>
      </Server>
```

Edit the global deployment descriptor conf/web.xml. Make sure that the invoker servlet is declared.

```
      <servlet>
         <servlet-name>invoker</servlet-name>
         <servlet-class>
          org.apache.catalina.servlets.InvokerServlet
         </servlet-class>
         <init-param>
            <param-name>debug</param-name>
            <param-value>0</param-value>
         </init-param>
         <load-on-startup>2</load-on-startup>
      </servlet>
```

Meanwhile, uncomment the servlet-path element that maps the path /servlet/* to the invoker servlet. After removing comment tags, the entry should look like the following:

```
         <!-- The mapping for the invoker servlet -->
         <servlet-mapping>
```

```
        <servlet-name>invoker</servlet-name>
        <url-pattern>/servlet/*</url-pattern>
    </servlet-mapping>
```

6.1.5 PostgreSQL Installation

PostgreSQL is the database system we choose in ODRS. It is included in RedHat by default. The reason that we choose PostgreSQL as ODRs's database system is because it provides a JDBC driver from which Java program easily connect database.

1. The first thing we need to check if the PostgreSQL is already in the operating system. Using the command to check if PostgreSQL exist in the operating system:

```
rpm -q postgresql
```

If PostgreSQL is not installd in the operating system, then use rpm to install it.

2. To install the RPMs just downloaded and run the RPM command. Installation of RPMs results in the creation of a user called postgres. To configure the database server, super user into postgres, and run the initdb command.

```
su postgres
initdb -D /var/lib/pgsql/data
```

Set an environmental variable PGDATA to point to the data directory. As root, add the following commands to

```
/etc/profile.d/myenv.sh:
```

```
PGDATA=/var/lib/pgsql/data
export PGDATA
```

3. Enable TCP connections to the database server, sothat JDBC will work. As user postgres, add the following line in the file /var/lib/pgsql/data/postgresql.conf.

```
tcpip_socket = true
```

4. Configure the system to run the postgresql server at boot time. Run the following as root:

```
/sbin/chkconfig --level 3 postgresql on
```

As root, start the server without rebooting:

```
/sbin/service postgresql start
```

5. Create database users. As postgres, run the following command, setting the password and answering no to the questions of whether the user can create databases or create new users.

```
createuser -P odrs
createdb -O odrs odrs
```

After having all the steps above executed, the database system is ready to go.

6.1.6 JAVA Database Connectivity (JDBC)

The API used to execute SQL statements is different for each database engine. Java programmers, however, are lucky and are freed from such database portability issues. They have a single API, the Java Database Connectivity API

(JDBC), that's portable between database engines. The JDBC library provides an interface for executing SQL statements. It provides the basic functionality for data access. http://jdbc.postgresql.org/download/. Download pg73jdbc3.jar and copy the file to /usr/java/jakarta-tomcat-4.1.29/common/lib/.

6.2 Olympiad Delegation Registration System Installation/Migration

The files making up ODRS application are organised into various sub-directories. Top-Level File Structure includes context.xml file, build.xml, log file, src and web directory. The src directory contains all the source code required to create a working application. This includes all the servlet and other .java files and all the test code. The sub-directories contain all the source code of the application. The classes subdirectory contains all the .java source files. The WEB-INF/classes subdirectories contains the .class files in their package directory structure (to work properly, a servlet must be in a named package). WEB-INF also contains the web.xml file, which configures the web application.

For ODRS, all the JSP and html files are stored in /olympiad/web/noc. All the classes are stored in /olympiad/web/WEB-INF/classes/olympiad. All the java files

are stored in /olympiad/src/olympiad/. And place the web.xml for ODRS in /Olympiad/web/WEB-INF/.

6.3 Backup and Restore

Backup is a particulary important action to maintain a system running manually. It represents complete solution for protection of important data and documents from any computer disaster: hard drive corruptions, operating system failure, programs dysfunctions. In case of ODRS server failure data can be easily recovered. There are two steps to back up ODRS. One is to backup the system files. The other step is to backup the database which is used by ODRS.

6.3.1 System Backup

All ODRS system files are located in the directory "/olympiad" and its subdirectories. Thus, to backup the ODRS system files, we need to backup the files in "/olympiad" directory. The method used here is by the compress program "tar".

```
tar -cf olympiad.tar /olympiad
```

It can compress the directory of "/olympiad" including its subdirectories to a file archive for which we would restore the ODRS system in the future.

6.3.2 Database Backup

To backup the database system, we use pg_dump command to backup the database used by ODRS. The following command is used to backup the database:

```
pg_dump olympiad | gzip > olympiad.zip
```

After executing the backup command above, the file olympiad.zip would be the backup file of the database.

6.3.3 System Restore

To restore the ODRS system file, simply extract the backup file by using the following command:

```
tar -xvf olympiad.tar /
```

By running command above, all ODRS system files will restore into the directory /olympiad and related to subdirectory as the same as before /olympiad backup.

6.3.4 Database Restore

To restore the database needed for the , go to the directory where your databse backup file is in, and execute the following commands:

```
createdb olympiad
gunzip -c olympiad.zip | psql olympiad
```

After the commands are executed, the database is restored to the database system.

CHAPTER SEVEN

CONCLUSION AND FUTURE DIRECTIONS

7.1 Conclusion and Future Directions

ODRS is a Web application system. It provides an efficient communication environment for the National Olympic Committees and the Organining Committee of Olympic Games to manage registrations of national delegates. NOC users can access the Olympiad registration around the world with the system. The main advantage of ODRS is that the system can automicaly check the registration of qualifications for each NOC during registration. Meanwhile NOC users can modify the delegate registration record at any time before the deadline of Olympiad registration. For the OCOG users, they can view information of all NOC registrations and get statistical data related to those information immidiatelly. Thus, ODRS is a practical and efficient web application for solving Olympiad registration issues.

ODRS is designed to implement all Olympic competitive event registration, including various officials delegation, for NOCs to participant in the Olympiad. In the future we can implement other event registrations rather than limiting to archery, basketball and officials. Because the rule of Olympiad registration has the similarity in the each time

Olympiad, we can slightly modify the ODRS in order that ODRS can be resued in the next Olympiad. Also, because ODRS includes the main sporting events of the world, it can be used in the individual world championship or other international tournaments with some modification.

On the other hand, since 1964 the results have been stored on computers, computer technology is now an established part of the management of the Olympics Games. In the contemporary Olympiad, there are special Games Management Systems Applications, which support a wide array of critical acticities, such as transportation management, sport entries and athlete qualification, medical encounters and accommodation etc. How to integrate the Web application of Olympiad registration into the "sport entries and athletes qualification" should be the main consideration of ODRS in the future.

APPENDIX A

OLYMPIAD DELEGATION REGISTRATION SYSTEM XML FILES PRINTOUT

```
Contex.xml
<Context    path="/olympiad"
            docBase="/home/xwang/olympiad/web"
            debug="0"
            reloadable="true"
            swallowOutput="true"
            useNaming="true">
<Logger     className="org.apache.catalina.logger.FileLogger"
            prefix=""
            suffix=".log"
            directory="/home/xwang/olympiad"
            timestamp="true"/>
<Resource   name="database"
            auth="Container"
            type="javax.sql.DataSource" />
<ResourceParams name="database">
          <parameter>
          <name>factory</name>
<value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
          </parameter>
          <parameter>
          <name>driverClassName</name>
          <value>org.postgresql.Driver</value>
          </parameter>
          <parameter>
          <name>url</name>
          <value>jdbc:postgresql://127.0.0.1/xwang</value>
          </parameter>
          <parameter>
          <name>username</name>
          <value>xwang</value>
          </parameter>
          <parameter>
          <name>password</name>
          <value>xwang</value>
          </parameter>
          <parameter>
          <name>maxActive</name>
          <value>4</value>
```

```
build.xml
<project basedir="." default="compile">
<!--.....-->
<!-- properties -->
<property   name="catalina.home"
            value="/usr/java/jakarta-tomcat-4.1.29" />
<property   name="servlet.jar"
value="${catalina.home}/common/lib/servlet.jar" />
<property   name="manager.jar"
value="${catalina.home}/server/lib/catalina-ant.jar" />
<property   name="commons-dbutils.jar"
value="/usr/java/commons-dbutils-1.0/commons-dbutils-1.0.jar" />
<property   name="commons-beanutils.jar"
value="/usr/java/commons-beanutils-1.6.1/commons-beanutils.jar" />
<property   name="jstl.home"
value="/usr/java/jakarta-taglibs-standard-1.0.4" />
<property   name="postgresql.jar"
value="/usr/share/pgsql/pg73b1jdbc3.jar" />
<!--.....-->
<!-- task definitions -->
<taskdef name="list"
            classname="org.apache.catalina.ant.ListTask"
            classpath="${manager.jar}" />
<taskdef name="install"
           classname="org.apache.catalina.ant.InstallTask"
            classpath="${manager.jar}" />
<taskdef name="remove"
            classname="org.apache.catalina.ant.RemoveTask"
            classpath="${manager.jar}" />
<!--.....-->
<!-- classpath -->
<path id="classpath">
<pathelement location="${servlet.jar}" />
<pathelement location="${commons-dbutils.jar}" />
</path>
<!--.....-->
<!-- minor targets -->
<target name="delete-logs">
   <delete>
   <fileset dir=".">
   <include name="*.log" />
   </fileset>
   </delete>
</target>
<target name="copy-libs">
   <!-- Copy tag library descriptor. -->
   <mkdir dir="web/WEB-INF/tld" />
   <copy todir="web/WEB-INF/tld">
   <fileset file="${jstl.home}/tld/c.tld" />
   </copy>
   <!-- Copy libraries. -->
   <mkdir dir="web/WEB-INF/lib" />
   <copy todir="web/WEB-INF/lib">
   <fileset file="${jstl.home}/lib/jstl.jar" />
   <fileset file="${jstl.home}/lib/standard.jar" />
   <fileset file="${commons-dbutils.jar}" />
   <fileset file="${commons-beanutils.jar}" />
```

```
        </copy>
      </target>
      <!--.....................................................-->
      <!-- main targets -->
      <target name="list" description="List the running web   apps.">
        <list   url="http://localhost:8080/manager" username="manager"
pas
        <mkdir dir="${catalina.home}/webapps/ROOT/docs"/>
        <mkdir dir="${catalina.home}/webapps/ROOT/docs/olympiad"/>
        <copy todir="${catalina.home}/webapps/ROOT/docs/olympiad" >
        <fileset dir="docs/html" />
        </copy>
      </target>
      <target name="clean" description="Delete all derived objects.">
        <delete dir="web/WEB-INF/classes" />
        <antcall target="delete-logs" />
      </target>
      <target name="compile" description="Compile Java source code.">
        <mkdir dir="web/WEB-INF/classes" />
        <!-- compile -->
        <javac  srcdir="src"
            destdir="web/WEB-INF/classes"
            fork="no"
            classpathref="classpath" />
      </target>
      <target name="restart" description="Restart web app.">
            <antcall target="stop" />
            <antcall target="start" />
      </target>
      <target name="start" depends="compile" description="start web
app.">
            <antcall target="delete-logs" />
            <install url="http://localhost:8080/manager"
            username="manager"
            password="manager"
            path="/olympiad"
            config="file://${basedir}/context.xml" />
      </target>
      <target name="stop" description="Stop web app.">
            <remove  url="http://localhost:8080/manager"
            username="manager"
            password="manager"
            path="/olympiad" />
            <antcall target="delete-logs" />
      </target>
      <target name="createdb" depends="compile">
           <java classname="beans.CreateDB">
           <classpath>
           <pathelement path="web/WEB-INF/classes"/>
           <pathelement location="${postgresql.jar}"/>
           </classpath>
           </java>
      </target>
      <target name="javadoc" depends="compile">
          <mkdir dir="${catalina.home}/webapps/ROOT/docs"/>
          <mkdir dir="${catalina.home}/webapps/ROOT/docs/olympiad"/>
```

```
          <mkdir
dir="${catalina.home}/webapps/ROOT/docs/olympiad/javadocs"/>
          <javadoc sourcepath="src" destdir="${catalina.home}
/webapps/ROOT/docs/olympiad/javadocs"
          packagenames="*"
          Private="true"
          Overview="${basedir}/docs/overview.html">
          <classpath refid="classpath" />
      <link href="http://web5.ias.csusb.edu:8080/tomcat-
docs/servletapi/" />
      <link href="http://web5.ias.csusb.edu:8080/commons-fileupload/" />
      <link href="http://java.sun.com/j2se/1.4.1/docs/api/" />
         </javadoc>
      </target>
      </project>
```

APPENDIX B

DELEGATION REGISTRATION SYSTEM

JAVA FILES PRINTOUT

```
package olympiad;


import javax.servlet.http.*;
import javax.naming.*;
import java.sql.*;
import javax.sql.DataSource;
import java.util.Vector;

public class Archery
{
   static private DataSource ds;

   static
   {
      try {
         InitialContext ic = new InitialContext();
         Context tomcatContext = (Context)
ic.lookup(Constants.jndiContainerContext);
         ds = (DataSource)
tomcatContext.lookup(Constants.jndiDatabaseName);
         if (ds == null) throw new RuntimeException("no
DataSource");
      } catch (Exception e) {
         throw new RuntimeException(e);
      }
   }

   private String idNumber, nocId, eventId, givenName, familyName,
               month, day, year, gender;
   private String[] events;
   private String ifid, hander, individual, rank, team, teamrank,
type, mqs;

   /*
    *@ For the archery athlete registration use
   */
   public Archery( String idNumber, String nocId, String eventId,
String familyName,
                  String givenName, String month, String day, String
year,
                       String gender, String []events )
   {
      this.idNumber = idNumber;
      this.nocId = nocId;
      this.eventId = eventId;
            this.familyName = familyName;
      this.givenName = givenName;
            this.month = month;
            this.day = day;
            this.year = year;
      this.gender = gender;
      this.events = events;
   }

  /*
   *@ For the view archery athlete list
```

```
        */
        public Archery( String idNumber, String nocId, String eventId,
String familyName,
                        String givenName, String month, String day, String
year,
                            String gender, String ifid, String hander,
String individual,
                        String rank, String team, String teamrank, String
type, String mqs )
        {
           this.idNumber = idNumber;
           this.nocId = nocId;
           this.eventId = eventId;
                 this.familyName = familyName;
           this.givenName = givenName;
                 this.month = month;
                 this.day = day;
                 this.year = year;
           this.gender = gender;
           this.ifid = ifid;
           this.hander = hander;
           this.individual = individual;
           this.rank = rank;
           this.team = team;
           this.teamrank = teamrank;
           this.type = type;
             this.mqs = mqs;
        }

        public String getIdNumber() { return idNumber; }
        public String getNocId() { return nocId; }
        public String getEventId() { return eventId; }
        public String getFamilyName() { return familyName; }
        public String getGivenName() { return givenName; }
        public String getMonth() { return month; }
        public String getDay() { return day; }
        public String getYear() { return year; }
        public String getGender() { return gender; }
        public String [] getEvents() {return events; }

        public String getIfid() {return ifid;}
        public String getHander() {return hander;}
        public String getIndividual() {return individual;}
        public String getRank() {return rank;}
        public String getTeam() {return team;}
        public String getTeamrank() {return teamrank;}
        public String getType() {return type;}
        public String getMqs() {return mqs;}
        /*
         * For an individual athlete, insert the registeration data
once submited.
         */
        public void insertDelegate(Archery archery) throws Exception
        {
           Connection connection = null;
                 try {
                connection = ds.getConnection();
```

```
                Statement statement = connection.createStatement();

              String s = "INSERT INTO athletes (id, noc, event,
familyName, givenName, " +
                          "month, day, year, gender)" +
                         "VALUES ('" + idNumber + "', '" + nocId + "', '"
+ eventId + "', '" +
                             familyName + "', '" + givenName + "', '" +
                         month + "', '" +  day + "', '" +  year + "', '"
+ gender + "' )";
                statement.executeUpdate(s);

              // for( int i=0; i<events.length; i++ ){
               String e = "INSERT INTO archery (id, ifid, rank, hander,
mqs, type, " +
                                                  "team, teamrank,
individual)" +
                           "VALUES ('" + idNumber + "', '" + events[0] +
"', '" + events[1] + "', '" +
                                events[2] + "', '" + events[3] + "', '" +
events[4] + "', '" +
                                events[5] + "', '" + events[6] + "', '" +
events[7] +  "'  );";
                  statement.executeUpdate(e);
                 }
              catch(SQLException sqle) {
                    System.out.println("Archery class insertDelegate
method - SQL Exception error:" + sqle);
              }
              catch(Exception e) {
                    System.out.println("Archery class, insertDelegate
method - Exception: " + e);
              }
              finally {
                 try{
                  connection.close();
                 }catch(SQLException sql){}
            }
          }//end insertDelegate

         public static Archery findDelegate(String id) throws Exception
         {
            Archery archery = null;
            Connection connection = null;
            try {
               connection = ds.getConnection();
               Statement statement = connection.createStatement();
               String s = "select * from athletes where id ='" + id +
"';";
               ResultSet rs = statement.executeQuery(s);
               if (!rs.first()) return null;
               String noc = rs.getString("noc");
               String event = rs.getString("event");
             String familyName = rs.getString("familyname");
               String givenName = rs.getString("givenname");
               String month = rs.getString("month");
               String day = rs.getString("day");
```

```
            String year = rs.getString("year");
            String gender = rs.getString("gender");

            s = "select * from archery where id = '" + id + "';";
            rs = statement.executeQuery(s);
          if (!rs.first()) return null;
          String ifid = rs.getString("ifid");
          String hander = rs.getString("hander");
          String individual = rs.getString("individual");
          String rank = rs.getString("rank");
          String team = rs.getString("team");
          String teamrank = rs.getString("teamrank");
          String type = rs.getString("type");
          String mqs = rs.getString("mqs");
            archery = new Archery(id, noc, event, familyName,
givenName,
                                  month, day, year, gender, ifid,
hander,
                              individual, rank, team, teamrank, type,
mqs);
        }
        catch(SQLException sqle) {
          System.out.println("ModifyDelete class, findId method -
SQL Exception error:" + sqle);
        }
        catch(Exception e) {
          System.out.println("ModifyDelete class, findId method -
Exception: " + e);
        }
        finally {
          connection.close();
        }
        return archery;
      }


      public static void deleteDelegate (String id) throws Exception
      {
        Connection connection = null;

        try {
          connection = ds.getConnection();
          Statement statement = connection.createStatement();
          //save id for feature adding other delegates in the same
noc
          String s = "insert into idstore (id) values ('" + id +
"');";
          statement.executeUpdate(s);

          s = "delete from athletes where id = '" + id + "';";
          statement.executeUpdate(s);
          s = "delete from archery where id = '" + id + "';";
          statement.executeUpdate(s);
        }
        catch(SQLException sqle) {
          System.out.println("Archery class, deleteDelegate method -
SQL Exception error:" + sqle);
```

```
            }
            catch(Exception e) {
              System.out.println("Archery class, deleteDelegate method -
Exception: " + e);
            }
            finally {
              connection.close();
            }
          }

          //for modify the archery delegate record
          public static void updateDelegate(Archery delegate) throws
Exception
          {
             String id = delegate.getIdNumber();
             String familyName = delegate.getFamilyName();
             String givenName = delegate.getGivenName();
             String month = delegate.getMonth();
             String day = delegate.getDay();
             String year =  delegate.getYear();
             String gender = delegate.getGender();
             String [] events = delegate.getEvents();

             Connection connection = null;
             try {
               connection = ds.getConnection();
               Statement statement = connection.createStatement();
               //update the archery delegate records in athletes table
parts
               String s = "update athletes set " +
                           "familyname = '" + familyName + "', " +
                           "givenname = '" + givenName + "', " +
                           "month = '" + month + "', " +
                           "day = '" + day + "', " +
                           "year = '" + year + "', " +
                           "gender = '" + gender + "'" +
                           "where id = '" + id + "'";
               statement.executeUpdate(s);

               s = "update archery set " +
                        "ifid = '" + events[0] + "', " +
                        "rank = '" + events[1] + "', " +
                        "hander = '" + events[2] + "', " +
                        "mqs = '" + events[3] + "', " +
                            "type = '" + events[4] + "', " +
                        "team = '" + events[5] + "', " +
                            "teamrank = '" + events[6] + "'" +
                            "where id = '" + id + "'";
               statement.executeUpdate(s);
               }
               catch(SQLException sqle) {
                 System.out.println("Archery class updateDelegate method
- SQL Exception error:" + sqle);
               }
               catch(Exception e) {
                 System.out.println("Archery class updateDelegate method
- Exception: " + e);
```

```
        }
        finally {
          connection.close();
      }
    }

}
```

```
package olympiad;


import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;
import java.util.Enumeration;


public class ArcheryServlet extends HttpServlet
{
   RequestDispatcher archeryPage;
   RequestDispatcher loginPage;
   RequestDispatcher viewListPage;

   public void init() throws ServletException
   {
        ServletContext context = getServletContext();

        archeryPage =
context.getRequestDispatcher(Constants.archeryPagePath);
        loginPage =
context.getRequestDispatcher(Constants.loginPagePath);
        viewListPage =
context.getRequestDispatcher(Constants.viewListPagePath);

        if (archeryPage == null) {
             throw new ServletException(Constants.archeryPagePath
+ " not found");
            }

        if (loginPage == null) {
             throw new ServletException(Constants.loginPagePath +
" not found");
            }

        if (viewListPage == null) {
             throw new ServletException(Constants.viewListPagePath
+ " not found");
            }
    }

    protected void doPost(HttpServletRequest request,
HttpServletResponse response)
        throws ServletException, IOException
       {
        System.out.println("ArcheryServlet running");

        String action = request.getParameter("action");
          if ( action.equals("cancel")) {
            viewListPage.forward(request, response);
            return;
          }

        String familyName = request.getParameter("familyName");
              if (familyName == null || familyName.equals("")) {
```

```
                request.setAttribute("error", "The familyName is a
required.");
                archeryPage.forward(request, response);
              return;
                }
            System.out.println("familyName is " + familyName);

              String givenName = request.getParameter("givenName");
            if (givenName == null || givenName.equals("")) {
                System.out.println("given name is null");
                request.setAttribute("error", "The givenName is a
required.");
                archeryPage.forward(request, response);
                return;
            }
            System.out.println("givenName is " + familyName);

            String month = request.getParameter("month");
                 if (month == null || month.equals("")) {
                request.setAttribute("error", "The month of birthday is
a required.");
                archeryPage.forward(request, response);
                return;
                 }

            String day = request.getParameter("day");
            if (day == null || day.equals("")) {
                request.setAttribute("error", "The day of birthday is a
required.");
                archeryPage.forward(request, response);
                return;
                 }

            String year =  request.getParameter("year");
                 if (year == null || year.equals("")) {
                request.setAttribute("error", "The year of birthday is
a required.");
                archeryPage.forward(request, response);
                return;
                 }

            String gender = request.getParameter("gender");
                 if (gender == null || gender.equals("")) {
                request.setAttribute("error", "Gender is a required.");
                archeryPage.forward(request, response);
                return;
                 }

            String events[] = request.getParameterValues("events");
            if(events == null || events.length == 0){
                request.setAttribute("error", "The event(s) is a
required.");
                archeryPage.forward(request, response);
                return;
            }
            else{
               for(int i=0; i<events.length; i++){
```

```
                System.out.println(events[i] + ",");
                }
            }

            String idNumber = "", nocId = "", eventId = "", nocName="";
            String eventName = "ar";

            eventId = EventId.archery;

            HttpSession session = request.getSession();
            if (session != null) {
               nocName = (String) session.getAttribute("noc");
               if (nocName == null){
                  request.setAttribute("error", "Please login again.");

                    response.sendRedirect(request.getContextPath() +
Constants.loginPagePath);
                  return;
                }
               System.out.println( "The session attribute is " + nocName
);
               NocId nocid = new NocId(nocName);
                 nocId = nocid.getNocCode();
               System.out.println( "The nocid is " + nocId );
            }

            IdNumber idnumber = new IdNumber(nocId, eventId, gender );

            try {
                idNumber = idnumber.getIdNumber();
            } catch (Exception e) {
                throw new ServletException(e.toString());
                }

            Archery archery = new Archery(idNumber, nocName, eventName,
                  familyName, givenName, month, day, year, gender,
events);

              request.setAttribute("archery", archery);

            try {
               archery.insertDelegate(archery);
            } catch (Exception e) {
                throw new ServletException(e.toString());
                }

          request.setAttribute("error", "Register sucess, please
register other again.");
           viewListPage.forward(request, response);
         }//end dopost
      }
```

```
package olympiad;


import javax.servlet.http.*;
import javax.naming.*;
import java.sql.*;
import javax.sql.DataSource;
import java.util.Vector;

public class Basketball
{
   static private DataSource ds;

   static
   {
      try {
         InitialContext ic = new InitialContext();
         Context tomcatContext = (Context)
ic.lookup(Constants.jndiContainerContext);
         ds = (DataSource)
tomcatContext.lookup(Constants.jndiDatabaseName);
         if (ds == null) throw new RuntimeException("no
DataSource");
      } catch (Exception e) {
         throw new RuntimeException(e);
      }
   }

   private String idNumber, nocId, eventId, givenName, familyName,
               month, day, year, gender;
   private String[] events;
   private String ifid, unifno, position, unifco, type;

   /*
    *@ For the basketball athlete registration use
   */
   public Basketball( String idNumber, String nocId, String
eventId, String familyName,
                  String givenName, String month, String day, String
year,
                        String gender, String []events )
   {
      this.idNumber = idNumber;
      this.nocId = nocId;
      this.eventId = eventId;
            this.familyName = familyName;
      this.givenName = givenName;
            this.month = month;
            this.day = day;
            this.year = year;
      this.gender = gender;
      this.events = events;
   }

  /*
   *@ For the view basketball athlete list
   */
```

```
        public Basketball( String idNumber, String nocId, String
eventId, String familyName,
                        String givenName, String month, String day, String
year,
                            String gender, String ifid, String unifno,
String position,
                        String unifco, String type )
        {
           this.idNumber = idNumber;
           this.nocId = nocId;
           this.eventId = eventId;
                 this.familyName = familyName;
           this.givenName = givenName;
                 this.month = month;
                 this.day = day;
                 this.year = year;
           this.gender = gender;
           this.ifid = ifid;
           this.unifno = unifno;
           this.position = position;
           this.unifco = unifco;
           this.type = type;
        }

        public String getIdNumber() { return idNumber; }
        public String getNocId() { return nocId; }
        public String getEventId() { return eventId; }
        public String getFamilyName() { return familyName; }
        public String getGivenName() { return givenName; }
        public String getMonth() { return month; }
        public String getDay() { return day; }
        public String getYear() { return year; }
        public String getGender() { return gender; }
        public String [] getEvents() {return events; }

        public String getIfid() {return ifid;}
        public String getUnifno() {return unifno;}
        public String getPosition() {return position;}
        public String getUnifco() {return unifco;}
        public String getType() {return type;}

        /*
         * For an individual athlete, insert the registeration data
once submited.
         */
        public void insertDelegate(Basketball basketball) throws
Exception
        {
           Connection connection = null;
                 try {
                connection = ds.getConnection();
                Statement statement = connection.createStatement();

              String s = "INSERT INTO athletes (id, noc, event,
familyName, givenName, " +
                           "month, day, year, gender)" +
```

```
                        "VALUES ('" + idNumber + "', '" + nocId + "', '"
+ eventId + "', '" +
                            familyName + "', '" + givenName + "', '" +
                        month + "', '" +  day + "', '" +  year + "', '"
+ gender + "' )";
                statement.executeUpdate(s);

             // for( int i=0; i<events.length; i++ ){
              String e = "INSERT INTO basketball (id, ifid, unifno,
position, unifco, type)" +
                          "VALUES ('" + idNumber + "', '" + events[0] +
"', '" + events[1] + "', '" +
                              events[2] + "', '" + events[3] + "', '" +
events[4] + "')";
                  statement.executeUpdate(e);
                 }
             catch(SQLException sqle) {
                   System.out.println("Basketball class insertDelegate
method - SQL Exception error:" + sqle);
             }
             catch(Exception e) {
                   System.out.println("Basketball class, insertDelegate
method - Exception: " + e);
             }
             finally {
                try{
                 connection.close();
                }catch(SQLException sql){}
           }
         }//end insertDelegate

        public static Basketball findDelegate(String id) throws
Exception
        {
           Basketball basketball = null;
           Connection connection = null;
           try {
              connection = ds.getConnection();
              Statement statement = connection.createStatement();
              String s = "select * from athletes where id ='" + id +
"';";
              ResultSet rs = statement.executeQuery(s);
              if (!rs.first()) return null;
              String noc = rs.getString("noc");
              String event = rs.getString("event");
            String familyName = rs.getString("familyname");
              String givenName = rs.getString("givenname");
              String month = rs.getString("month");
              String day = rs.getString("day");
              String year = rs.getString("year");
              String gender = rs.getString("gender");

              s = "select * from basketball where id = '" + id + "';";
              rs = statement.executeQuery(s);
            if (!rs.first()) return null;
            String ifid = rs.getString("ifid");
            String unifno = rs.getString("unifno");
```

```
            String position = rs.getString("position");
            String unifco = rs.getString("unifco");
            String type = rs.getString("type");
              basketball = new Basketball(id, noc, event, familyName,
givenName,
                          month, day, year, gender, ifid, unifno,
position, unifco, type);
            }
            catch(SQLException sqle) {
               System.out.println("Basketball class, findId method - SQL
Exception error:" + sqle);
            }
            catch(Exception e) {
               System.out.println("Basketball class, findId method -
Exception: " + e);
            }
            finally {
               connection.close();
            }
            return basketball;
         }


         public static void deleteDelegate (String id) throws Exception
         {
            Connection connection = null;

            try {
              connection = ds.getConnection();
              Statement statement = connection.createStatement();
              //save id for feature adding other delegates in the same
noc
              String s = "insert into idstore (id) values ('" + id +
"');";
              statement.executeUpdate(s);

              s = "delete from athletes where id = '" + id + "';";
              statement.executeUpdate(s);
              s = "delete from basketball where id = '" + id + "';";
              statement.executeUpdate(s);
            }
            catch(SQLException sqle) {
              System.out.println("Basketball class, deleteDelegate
method - SQL Exception error:" + sqle);
            }
            catch(Exception e) {
              System.out.println("Basketball class, deleteDelegate
method - Exception: " + e);
            }
            finally {
              connection.close();
            }
         }

         //for modify the basketball delegate record
         public static void updateDelegate(Basketball delegate) throws
Exception
```

```
        {
            String id = delegate.getIdNumber();
            String familyName = delegate.getFamilyName();
            String givenName = delegate.getGivenName();
            String month = delegate.getMonth();
            String day = delegate.getDay();
            String year =  delegate.getYear();
            String gender = delegate.getGender();
            String [] events = delegate.getEvents();

            Connection connection = null;
            try {
              connection = ds.getConnection();
              Statement statement = connection.createStatement();
              //update the basketball delegate records in athletes table
parts
              String s = "update athletes set " +
                          "familyname = '" + familyName + "', " +
                          "givenname = '" + givenName + "', " +
                          "month = '" + month + "', " +
                          "day = '" + day + "', " +
                          "year = '" + year + "', " +
                          "gender = '" + gender + "'" +
                          "where id = '" + id + "'";
              statement.executeUpdate(s);

              s = "update basketball set " +
                       "ifid = '" + events[0] + "', " +
                       "unifno = '" + events[1] + "', " +
                       "position = '" + events[2] + "', " +
                       "unifco = '" + events[3] + "', " +
                           "type = '" + events[4] + "'" +
                           "where id = '" + id + "'";
              statement.executeUpdate(s);
              }
              catch(SQLException sqle) {
                System.out.println("Basketball class updateDelegate
method - SQL Exception error:" + sqle);
              }
              catch(Exception e) {
                System.out.println("Basketball class updateDelegate
method - Exception: " + e);
              }
              finally {
                connection.close();
            }
        }

    }
```

```
package olympiad;


import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;
import java.util.Enumeration;


public class BasketballServlet extends HttpServlet
{
   RequestDispatcher basketballPage;
   RequestDispatcher loginPage;
   RequestDispatcher viewListPage;

   public void init() throws ServletException
   {
        ServletContext context = getServletContext();

        basketballPage =
context.getRequestDispatcher(Constants.basketballPagePath);
        loginPage =
context.getRequestDispatcher(Constants.loginPagePath);
        viewListPage =
context.getRequestDispatcher(Constants.viewListPagePath);

      if (basketballPage == null) {
            throw new
ServletException(Constants.basketballPagePath + " not found");
            }

      if (loginPage == null) {
            throw new ServletException(Constants.loginPagePath +
" not found");
            }

      if (viewListPage == null) {
            throw new ServletException(Constants.viewListPagePath
+ " not found");
            }
   }

   protected void doPost(HttpServletRequest request,
HttpServletResponse response)
      throws ServletException, IOException
     {
      System.out.println("Basketball Servlet running");

      String action = request.getParameter("action");
        if ( action.equals("cancel")) {
          viewListPage.forward(request, response);
          return;
        }
      System.out.println("BasketballServlet action is " + action);

      String familyName = request.getParameter("familyName");
            if (familyName == null || familyName.equals("")) {
```

```
                request.setAttribute("error", "The familyName is a
required.");
                basketballPage.forward(request, response);
              return;
                }
            System.out.println("familyName is " + familyName);

              String givenName = request.getParameter("givenName");
            if (givenName == null || givenName.equals("")) {
                System.out.println("given name is null");
                request.setAttribute("error", "The givenName is a
required.");
                basketballPage.forward(request, response);
                return;
            }
            System.out.println("givenName is " + familyName);

            String month = request.getParameter("month");
                  if (month == null || month.equals("")) {
                request.setAttribute("error", "The month of birthday is
a required.");
                basketballPage.forward(request, response);
                return;
                }

            String day = request.getParameter("day");
            if (day == null || day.equals("")) {
                request.setAttribute("error", "The day of birthday is a
required.");
                basketballPage.forward(request, response);
                return;
                }

            String year =  request.getParameter("year");
                  if (year == null || year.equals("")) {
                request.setAttribute("error", "The year of birthday is
a required.");
                basketballPage.forward(request, response);
                return;
                }

            String gender = request.getParameter("gender");
                  if (gender == null || gender.equals("")) {
                request.setAttribute("error", "Gender is a required.");
                basketballPage.forward(request, response);
                return;
                }

            String events[] = request.getParameterValues("events");
            if(events == null || events.length == 0){
                request.setAttribute("error", "The event(s) is a
required.");
                basketballPage.forward(request, response);
                return;
            }
            else{
               for(int i=0; i<events.length; i++){
```

```
                System.out.println(events[i] + ",");
                }
            }

            String idNumber = "", nocId = "", eventId = "", nocName="";
            String eventName = "bk";

            eventId = EventId.basketball;

            HttpSession session = request.getSession();
            if (session != null) {
               nocName = (String) session.getAttribute("noc");
               if (nocName == null){
                  request.setAttribute("error", "Please login again.");

                    response.sendRedirect(request.getContextPath() +
Constants.loginPagePath);
                  return;
                }
               System.out.println( "The session attribute is " + nocName
);
               NocId nocid = new NocId(nocName);
                 nocId = nocid.getNocCode();
               System.out.println( "The nocid is " + nocId );
            }

            IdNumber idnumber = new IdNumber(nocId, eventId, gender );

            try {
                idNumber = idnumber.getIdNumber();
            } catch (Exception e) {
                throw new ServletException(e.toString());
                }

            Basketball basketball = new Basketball(idNumber, nocName,
eventName,
                  familyName, givenName, month, day, year, gender,
events);

              request.setAttribute("basketball", basketball);

            try {
               basketball.insertDelegate(basketball);
            } catch (Exception e) {
                throw new ServletException(e.toString());
                }

          request.setAttribute("error", "Register sucess, please
register other again.");
           viewListPage.forward(request, response);
         }//end dopost
      }
```

```
package olympiad;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;

public class ChangePasswordServlet extends HttpServlet
{
   RequestDispatcher changePasswordPage;
   RequestDispatcher nocPage;

   public void init() throws ServletException
   {
      ServletContext context = getServletContext();
      changePasswordPage =
context.getRequestDispatcher(Constants.changePasswordPagePath);
      if (changePasswordPage == null) {
         throw new
ServletException(Constants.changePasswordPagePath + " not found");
      }

      nocPage =
context.getRequestDispatcher(Constants.nocPagePath);
      if (nocPage == null) {
         throw new ServletException(Constants.nocPagePath + " not
found");
      }
   }

   protected void doPost(HttpServletRequest request,
HttpServletResponse response)
   throws ServletException, IOException
   {
      System.out.println( "Changepassword servlet running");
      String userid = request.getParameter("userid");
      if (userid == null || userid.equals("") ){
         request.setAttribute("error", "User ID is a required.");
         changePasswordPage.forward(request, response);
         return;
      }
      String password = request.getParameter("password");
      if (password == null) password = "";

      User user = null;
      try {
         user = User.find(userid);
      } catch (Exception e) {
         throw new ServletException(e.toString());
      }
      if (user == null) {
         request.setAttribute("error", "User ID doesn't exist.");
         changePasswordPage.forward(request, response);
         return;
      }
      if (!user.getPassword().equals(password)) {
         request.setAttribute("error", "Password incorrect.");
         changePasswordPage.forward(request, response);
```

```
            return;
         }

         String newuserid = request.getParameter("newuserid");
         if (newuserid == null || userid.equals("") ){
            request.setAttribute("error", "New user ID is a
required.");
            changePasswordPage.forward(request, response);
            return;
         }

         String newfirstpw = request.getParameter("newfirstpw");
         String newsecondpw = request.getParameter("newsecondpw");
         if ( !newfirstpw.equals(newsecondpw) ) {
         request.setAttribute("error", "the new passwords don't
match");
            changePasswordPage.forward(request, response);
            return;
         }

         try {
            User.update(userid, newuserid, newfirstpw);
         } catch (Exception e) {
            throw new ServletException(e.toString());
         }
         nocPage.forward(request, response);
      }
   }
```

```
package olympiad;

import java.util.Vector;

public class Constants
{
   public static final String jndiContainerContext =
"java:comp/env";
   public static final String jndiDatabaseName = "database";

   public static final String loginPagePath = "/login.jsp";
   public static final String changePasswordPagePath =
"/changepassword.jsp";
   public static final String ocogPagePath = "/ocog/ocog.jsp";
   public static final String sorryPagePath = "/sorry.jsp";
   public static final String nocPagePath = "/noc/noc.jsp";
   public static final String nocListPagePath =
"/noc/noclist.jsp";
   public static final String nocEventPagePath =
"/noc/nocevent.jsp";

   /*
    *under noc subdirectory for adding new delegate jsp files
    */
   public static final String officialPagePath =
"/noc/official.jsp";
   public static final String archeryPagePath =
"/noc/archery.jsp";
   public static final String athleticPagePath =
"/noc/athletic.jsp";
   public static final String badmintonPagePath =
"/noc/badminton.jsp";
   public static final String baseballPagePath =
"/noc/baseball.jsp";
   public static final String basketballPagePath =
"/noc/basketball.jsp";
   public static final String boxingPagePath = "/noc/boxing.jsp";
   public static final String canoePagePath = "/noc/canoe.jsp";
   public static final String cyclingPagePath =
"/noc/cycling.jap";
   public static final String equestrianPagePath =
"/noc/equestrian.jsp";
   public static final String fencingPagePath = "/noc/fecing.jsp";
   public static final String footballPagePath =
"/noc/football.jsp";
   public static final String gymnasticsPagePath =
"/noc/gymnastics.jsp";
   public static final String handballPagePath =
"/noc/handball.jsp";
   public static final String hockeyPagePath = "/noc/hockey.jsp";
   public static final String judoPagePath = "/noc/judo.jsp";
   public static final String pentathonPagePath =
"/noc/pentathon.jsp";
   public static final String rowingPagePath = "/noc/rowing.jsp";
   public static final String softballPagePath =
"/noc/softball.jsp";
```

```
        public static final String sailingPagePath =
"/noc/sailing.jsp";
        public static final String shootingPagePath =
"/noc/shooting.jsp";
        public static final String swimmingPagePagePath =
"/noc/swimming.jsp";
        public static final String triathlonPath =
"/noc/triathlon.jsp";
        public static final String tabletennesPagePath =
"/noc/tabletennes.jsp";
        public static final String taekwondoPagePath =
"/noc/taekwondopath.jsp";
        public static final String tennesPagePath = "/noc/tennis.jsp";
        public static final String volleyballPagePath =
"/noc/volleyball.jsp";
        public static final String wightliftingPagePath =
"/noc/wightlifting.jsp";
        public static final String wrestlingPagePath =
"/noc/wrestling.jsp";

        /*
         *under noc subdirectory for modify or delete existing delegate
jsp files
         */
        public static final String viewListPagePath =
"/noc/viewlist.jsp";
        public static final String viewOfficialPagePath =
"/noc/viewofficial.jsp";
        public static final String viewArcheryPagePath =
"/noc/viewarchery.jsp";
        public static final String viewBasketballPagePath =
"/noc/viewbasketball.jsp";
        public static final String modifyOfficialPagePath =
"/noc/modifyofficial.jsp";
        public static final String modifyArcheryPagePath =
"/noc/modifyarchery.jsp";
        public static final String modifyBasketballPagePath =
"/noc/modifybasketball.jsp";
        public static final String deleteOfficialPagePath =
"/noc/deleteofficial.jsp";
        public static final String deleteArcheryPagePath =
"/noc/deletearchery.jsp";
        public static final String deleteBasketballPagePath =
"/noc/deletebasketball.jsp";

    }
```

```
package olympiad;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;

public class DeleteArcheryServlet extends HttpServlet
{
   RequestDispatcher viewListPage;
   RequestDispatcher viewArcheryPage;

   public void init() throws ServletException
   {
      ServletContext context = getServletContext();

      viewListPage =
context.getRequestDispatcher(Constants.viewListPagePath);
      if (viewListPage == null) {
         throw new ServletException(Constants.viewListPagePath + "
not found");
      }

      viewArcheryPage =
context.getRequestDispatcher(Constants.viewArcheryPagePath);
      if (viewArcheryPage == null) {
         throw new ServletException(Constants.viewArcheryPagePath
+ " not found");
      }
   }

   protected void doGet(HttpServletRequest request,
HttpServletResponse response)
   throws ServletException, IOException
   {
      System.out.println("DeleteArcheryServlet is running");

      String id = request.getParameter("id");
      if (id == null) {
         request.setAttribute("error", "Request error: id value
missing.");
         viewArcheryPage.forward(request, response);
         return;
      }

      String action = request.getParameter("action");
      if (action == null) {
         request.setAttribute("error", "Request error: action
value missing.");
         viewArcheryPage.forward(request, response);
         return;
      }

      if (action.equals("delete")){
      try {
        Archery.deleteDelegate(id);
        } catch (Exception e) {
          throw new ServletException(e.toString());
```

```
            }
            request.setAttribute("error", "Delete sucess, try again");
            viewListPage.forward(request, response);
            return;
        }

        if (action.equals("cancel")){
            viewArcheryPage.forward(request, response);
            return;
        }
    }
}
```

```
package olympiad;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;

public class DeleteBasketballServlet extends HttpServlet
{
   RequestDispatcher viewListPage;
   RequestDispatcher viewBasketballPage;

   public void init() throws ServletException
   {
      ServletContext context = getServletContext();

      viewListPage =
context.getRequestDispatcher(Constants.viewListPagePath);
      if (viewListPage == null) {
         throw new ServletException(Constants.viewListPagePath + "
not found");
      }

      viewBasketballPage =
context.getRequestDispatcher(Constants.viewBasketballPagePath);
      if (viewBasketballPage == null) {
         throw new
ServletException(Constants.viewBasketballPagePath + " not found");
      }
   }

   protected void doGet(HttpServletRequest request,
HttpServletResponse response)
   throws ServletException, IOException
   {
      System.out.println("DeleteBasketballServlet is running");

      String id = request.getParameter("id");
      if (id == null) {
         request.setAttribute("error", "Request error: id value
missing.");
         viewBasketballPage.forward(request, response);
         return;
      }

      String action = request.getParameter("action");
      if (action == null) {
         request.setAttribute("error", "Request error: action
value missing.");
         viewBasketballPage.forward(request, response);
         return;
      }

      if (action.equals("delete")){
      try {
        Basketball.deleteDelegate(id);
        } catch (Exception e) {
          throw new ServletException(e.toString());
```

```
            }
            request.setAttribute("error", "Delete sucess, try again");
            viewListPage.forward(request, response);
            return;
        }

        if (action.equals("cancel")){
            viewBasketballPage.forward(request, response);
            return;
        }
    }
}
```

```
package olympiad;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;

public class DeleteOfficialServlet extends HttpServlet
{
   RequestDispatcher viewListPage;
   RequestDispatcher viewOfficialPage;

   public void init() throws ServletException
   {
      ServletContext context = getServletContext();

      viewListPage =
context.getRequestDispatcher(Constants.viewListPagePath);
      if (viewListPage == null) {
         throw new ServletException(Constants.viewListPagePath + "
not found");
      }

      viewOfficialPage =
context.getRequestDispatcher(Constants.viewOfficialPagePath);
      if (viewOfficialPage == null) {
         throw new ServletException(Constants.viewOfficialPagePath
+ " not found");
      }
   }

   protected void doGet(HttpServletRequest request,
HttpServletResponse response)
   throws ServletException, IOException
   {
      System.out.println("ViewOfficialServlet is running");

      String id = request.getParameter("id");
      if (id == null) {
         request.setAttribute("error", "Request error: id value
missing.");
         viewOfficialPage.forward(request, response);
         return;
      }

      String action = request.getParameter("action");
      if (action == null) {
         request.setAttribute("error", "Request error: action
value missing.");
         viewOfficialPage.forward(request, response);
         return;
      }

      if (action.equals("delete")){
      try {
        Official.deleteDelegate(id);
        } catch (Exception e) {
          throw new ServletException(e.toString());
```

```
            }
            request.setAttribute("error", "Delete sucess, try again");
            viewListPage.forward(request, response);
            return;
        }

        if (action.equals("cancel")){
            viewOfficialPage.forward(request, response);
            return;
        }
    }
}
```

```
package olympiad;


public class EventId
{
   public static final String official = "30";

   public static final String archery = "01";
   public static final String athletic = "02";
   public static final String badminton = "03";
   public static final String baseball = "04";
   public static final String basketball = "05";
   public static final String boxing = "06";
   public static final String canoe = "07";
   public static final String cycling = "08";
   public static final String equestrian = "09";
   public static final String fencing = "10";
   public static final String football = "11";
   public static final String gymnastics = "12";
   public static final String handball = "13";
   public static final String hockey = "14";
   public static final String judo = "15";
   public static final String pentathon = "16";
   public static final String rowing = "17";
   public static final String softball = "18";
   public static final String sailing = "19";
   public static final String shooting = "20";
   public static final String swimming = "21";
   public static final String triathlon = "22";
   public static final String tabletennes = "23";
   public static final String taekwondo = "24";
   public static final String tennes = "25";
   public static final String volleyball = "26";
   public static final String wightlifting = "27";
   public static final String wrestling = "28";


}
```

```
package olympiad;

public class Event
{

  public static final String ar = "archery";
  public static final String at = "athletics";
  public static final String bd = "badminton";
  public static final String bs = "baseball";
  public static final String bk = "basketball";
  public static final String bo = "boxing";
  public static final String ca = "canoe";
  public static final String cy = "cycling";
  public static final String eq = "equestrina";
  public static final String fe = "fencing";
  public static final String fo = "football";
  public static final String gy = "gymnastics";
  public static final String ha = "handball";
  public static final String ho = "hockey";
  public static final String ju = "judo";
  public static final String mp = "modern pentathlon";
  public static final String ro = "rowing";
  public static final String sa = "sailing";
  public static final String sh = "shooting";
  public static final String so = "softball";
  public static final String sw = "swimming";
  public static final String tt = "table tennis";
  public static final String ta = "taekwondo";
  public static final String te = "tennis";
  public static final String tr = "triathlon";
  public static final String vo = "volleyball";
  public static final String we = "weightlifting";
  public static final String wr = "wrestling";

  /*using ar substatud for artchey events*/
  public static final String ar01 = "individual";
  public static final String ar02 = "team";

  /*using at substatud for athletics events*/
  public static final String at01 = "100m";
  public static final String at02 = "200m";
  public static final String at03 = "400m";
  public static final String at04 = "800m";
  public static final String at05 = "1500m";
  public static final String at06 = "3000m";
  public static final String at07 = "5000m";
  public static final String at08 = "10000m";
  public static final String at09 = "100m hurdles"; //female
  public static final String at10 = "110m hurdles"; //male
  public static final String at11 = "400m hurdles";
  public static final String at12 = "3000m steeplechase"; //male
  public static final String at13 = "4x100m relay";
  public static final String at14 = "4x400m relay";
  public static final String at15 = "High Jump";
  public static final String at16 = "Pole Vault";
  public static final String at17 = "Long Jump";
  public static final String at18 = "Triple Jump";
```

```
        public static final String at19 = "Shot put";
        public static final String at20 = "Discus Throw";
        public static final String at21 = "Hammer Throw";
        public static final String at22 = "Javelin Throw";
        public static final String at23 = "Decathlon"; //male
        public static final String at24 = "Heptathlon"; //female
        public static final String at25 = "20Km Race Walk";
        public static final String at26 = "50Km Race walk"; //male
        public static final String at27 = "Matathon Race";

        /*using bd substatude for badminton events*/
        public static final String bd01 = "sigles";
        public static final String bd02 = "doubles";
        public static final String bd03 = "mixed doubles";

        /*using bs substatude for baseball event*/
        public static final String bs00 = "Team";

        /*using bk substatude for basketball event*/
        public static final String bk00 = "Team";

        /*using bo substatute for boxing events*/
        public static final String bo01 = "Light Fly Weight (-48Kg)";
        public static final String bo02 = "Fly Weight (51Kg)";
        public static final String bo03 = "Bantam Weight (54Kg)";
        public static final String bo04 = "Feather Weight (57Kg)";
        public static final String bo05 = "Light Weight (60Kg)";
        public static final String bo06 = "Light Welter Weight (64Kg)";
        public static final String bo07 = "Welter Weight (69Kg)";
        public static final String bo08 = "Middle Weight (75Kg)";
        public static final String bo09 = "Light Heavy Weight (81Kg)";
        public static final String bo10 = "Heavy Weight (91Kg)";
        public static final String bo11 = "Supter Heavy Weight (+91Kg)";


        public static final String ca01 = "Slalom Kayak Single";
        public static final String ca02 = "Slalom Canoe Single"; //male
        public static final String ca03 = "Slalom Canoe Double"; //male
        public static final String ca04 = "Flatwater Kayak Single 500m";
        public static final String ca05 = "Flatwater Kayak double 500m";
        public static final String ca06 = "Flatwater Kayak Single
1000m"; //male
        public static final String ca07 = "Flatwater Kayak Double
1000m";
        public static final String ca08 = "Flatwater Kayak Four 1000m";
        public static final String ca09 = "Flatwater Canoe Single 500m";
//male
        public static final String ca10 = "Flatwater Canoe Double 500m";
//male
        public static final String ca11 = "Flatwater Canoe Single
1000m"; //male
        public static final String ca12 = "Flatwater Canoe Double
1000m"; //male

        //Track cycling
        public static final String cy01 = "Individual Time Trial";
        public static final String cy02 = "1Km Time Trial"; //male
```

```
        public static final String cy03 = "500m Time Trial"; //female
        public static final String cy04 = "Sprint";
        public static final String cy05 = "Individual Pursuit";
        public static final String cy06 = "Points Race";
        public static final String cy07 = "Madison"; //male
        public static final String cy08 = "Team Pursuit"; //male
        public static final String cy09 = "Team Sprint"; //male
        public static final String cy10 = "Keirin"; //male
        //Road cycling
        public static final String cy11 = "Road Race";
        //Mountain bike
        public static final String cy12 = "Cross Country";


        public static final String eq01 = "Dressage";
        public static final String eq02 = "Jumping";
        public static final String eq03 = "Eventing";

        public static final String fe01 = "Individual Epee";
        public static final String fe02 = "Team Epee";
        public static final String fe03 = "Individual Foil";
        public static final String fe04 = "Team Foil"; //male
        public static final String fe05 = "Individual Sabre";
        public static final String fe06 = "Team Sabre"; //male

        public static final String fo01 = "Team";

        public static final String gy01 = "Individual";
        public static final String gy02= "Team";

        public static final String ha01 = "Team";

        public static final String ho01 = "Team";

        public static final String ju01 = "Extra Lightweight (-60Kg)";
//male
        public static final String ju02 = "Half Lightweight (-66Kg)";
//male
        public static final String ju03 = "Lightweight (-73Kg)"; //male
        public static final String ju04 = "Half Middleweight (-81Kg)";
//male
        public static final String ju05 = "Middleweight (-90Kg)"; //male
        public static final String ju06 = "Half Heavyweight (-100Kg)";
//male
        public static final String ju07 = "Heavyweight (+100Kg)"; //male
        public static final String ju08 = "Extra Lightweight (-48Kg)";
//female
        public static final String ju09 = "Half Lightweight (-52Kg)";
//female
        public static final String ju10 = "Lightweight (-57Kg)";
//female
        public static final String ju11 = "Half Middleweight (-63Kg)";
//female
        public static final String ju12 = "Middleweight (-70Kg)";
//female
        public static final String ju13 = "Half Heavyweight (-78Kg)";
//female
```

```
        public static final String ju14 = "Heavyweight (+78Kg)";
//female

        public static final String pe01 = "Individual";

        /*using "ro" substatute for rowing event*/
        public static final String ro01 = "Single Sculls";
        public static final String ro02 = "Double Sculls";
        public static final String ro03 = "Quadruple Sculls";
        public static final String ro04 = "Pair";
        public static final String ro05 = "Four"; //male
        public static final String ro06 = "Lightweight Double Sculls";
        public static final String ro07 = "Lightweight Fours"; //male
        public static final String ro08 = "Eight";

        /*using "sa" substatute for sailing event*/
        public static final String sa01 = "Windsufer (Mistral)";
        public static final String sa02 = "Single-handed Dinghy (Finn)";
//male
        public static final String sa03 = "Single-handed Dinghy Open
(Laser)";
        public static final String sa04 = "Single-handed Dinghy
(Europe)"; //female
        public static final String sa05 = "Double-handed Dinghy (740)";
        public static final String sa06 = "Double-handed Dinghy (49er)";
//male
        public static final String sa07 = "Double-handed Dinghy Open";
        public static final String sa08 = "Multhull Open (Tomado)";
        public static final String sa09 = "Keelboat (Star)"; //male
        public static final String sa10 = "Keelboat (yingling)";
//female

        /*using "sh" substatude for shooting event*/
        public static final String sh01 = "10m Running Target"; //male
        public static final String sh02 = "10m Air Rifle";
        public static final String sh03 = "10m Air Pistol";
        public static final String sh04 = "25m Rapid fire pistol";
//male
        public static final String sh05 = "25m Pistol"; //femal
        public static final String sh06 = "50m Rifle Prone"; //male
        public static final String sh07 = "50m Rifle 3 Positions";
        public static final String sh08 = "50m Pistol"; //male
        public static final String sh09 = "Trap";
        public static final String sh10 = "Skeet";
        public static final String sh11 = "Double";


        public static final String sw02 = "50m Freestyle";
        public static final String sw03 = "100m Freestyle";
        public static final String sw04 = "200m Freestyle";
        public static final String sw05 = "400m Freestyle";
        public static final String sw06 = "1500m Freestyle";
        public static final String sw07 = "100m Backstroke";
        public static final String sw08 = "200m Backstroke";
        public static final String sw09 = "100m Butterfly";
        public static final String sw10 = "200m Butterfly";
        public static final String sw11 = "100m Breaststroke";
```

```
public static final String sw12 = "200m Breaststroke";
public static final String sw13 = "200m Individual Medley";
public static final String sw14 = "400m Induvidual Medley";
public static final String sw15 = "4x100m Freestyle Relay";
public static final String sw16 = "4x200m Freestyle Relay";
public static final String sw17 = "4x100m Medley Relay";

public static final String tr01 = "Individual";

public static final String tt01 = "Single";
public static final String tt02 = "Double";

public static final String te01 = "Single";
public static final String te02 = "Double";

public static final String tk01 = "Under 58Kg"; //male
public static final String tk02 = "Under 68Kg"; //male
public static final String tk03 = "Under 80Kg"; //male
public static final String tk04 = "Over 80Kg"; //male
public static final String tk05 = "Under 49Kg"; //female
public static final String tk06 = "Under 57Kg"; //female
public static final String tk07 = "Under 67Kg"; //female
public static final String tk08 = "Over 67Kg"; //female

public static final String Vo01 = "Team";
public static final String Vo02 = "Beach Team";

public static final String We01 = "56Kg"; //male
public static final String We02 = "62Kg"; //male
public static final String We03 = "69Kg"; //male
public static final String We04 = "77Kg"; //male
public static final String We05 = "85Kg"; //male
public static final String We06 = "94Kg"; //male
public static final String We07 = "105Kg"; //male
public static final String We08 = "+105Kg"; //male
public static final String We09 = "48Kg"; //female
public static final String We10 = "53Kg"; //female
public static final String We11 = "58Kg"; //female
public static final String We12 = "63Kg"; //female
public static final String We13 = "69Kg"; //female
public static final String We14 = "75Kg"; //female
public static final String We15 = "+75Kg"; //female

public static final String wr01 = "55Kg Greco-Roman";
public static final String wr02 = "60Kg Greco-Roman";
public static final String wr03 = "66Kg Greco-Roman";
public static final String wr04 = "74Kg Greco-Roman";
public static final String wr05 = "84Kg Greco-Roman";
public static final String wr06 = "96Kg Greco-Roman";
public static final String wr07 = "120Kg Greco-Roman";
public static final String wr08 = "55Kg Freestyle";
public static final String wr09 = "60Kg Freestyle";
public static final String wr10 = "66Kg Freestyle";
public static final String wr11 = "74Kg Freestyle";
public static final String wr12 = "84Kg Freestyle";
public static final String wr13 = "96Kg Freestyle";
public static final String wr14 = "120Kg Freestyle";
```

```
  public static final String wr15 = "48Kg Woman";
  public static final String wr16 = "55Kg Woman";
  public static final String wr17 = "67Kg Woman";
  public static final String wr18 = "72Kg Woman";
}
```

```
package olympiad;


import java.util.Hashtable;
import javax.naming.*;
import java.sql.*;
import javax.sql.DataSource;
import java.io.IOException;


public class IdNumber
{
   static private DataSource ds;

   static
   {
      try {
         InitialContext ic = new InitialContext();
         Context tomcatContext = (Context)
ic.lookup(Constants.jndiContainerContext);
         ds = (DataSource)
tomcatContext.lookup(Constants.jndiDatabaseName);
         if (ds == null) throw new RuntimeException("no
DataSource");
      } catch (Exception e) {
         throw new RuntimeException(e);
      }
   }

   private String nocId, eventId, gender, delegate;

   public IdNumber(String nocId, String eventId, String gender)
   {
      this.eventId = eventId;
      this.nocId = nocId;
      if (gender.equals("m")) {this.gender = "0";}
      else {this.gender = "1";}
   }

   /**
    * @returns idnumber is id associate with the deletater added
    */
   public String getIdNumber() throws Exception
   {
      String idNumber = "";
      Connection connection = null;

      try {
        connection = ds.getConnection();
        Statement statement = connection.createStatement();
      String s = "select * from idstore where id like '" + nocId +
"%' ";
        ResultSet rs = statement.executeQuery(s);
      if (rs.first()) {
         String id = rs.getString("id");
         delegate = id.substring(6, 9);
           s = "delete from idstore where id = '" + id + "';";
```

```
                statement.executeUpdate(s);
            } else {
                s = "SELECT COUNT (*)  FROM athletes WHERE id LIKE '" +
nocId + "%' ";
                rs = statement.executeQuery(s);
                if (!rs.first()) return null;
                int athleteCount = rs.getInt( "count");
                s = "SELECT COUNT (*)  FROM officials WHERE id LIKE '"
+ nocId + "%' ";
                rs = statement.executeQuery(s);
                if (!rs.first()) return null;
              int officialCount = rs.getInt( "count");
                int count = athleteCount + officialCount;
                if (count < 9) { delegate = "00" + (count + 1); }
              if (count == 9) {delegate = "010";}
                if (9 < count && count < 99) { delegate = "0" + (count
+ 1); }
              if (count == 99) {delegate = "100";}
              if (99 < count) { delegate = "(count + 1)"; }
            }
            idNumber = nocId + eventId + gender + delegate;
            }catch(SQLException sqle) {
                System.out.println("IdNumber class, getIdNumber method -
SQL Exception error:" + sqle);
            }
            catch(Exception e) {
                System.out.println("IdNumber class, getIdNumber method -
Exception: " + e);
            }
            finally {
              try{
                connection.close();
              }catch(SQLException sql){}
            }
              System.out.println( "IdNumber class, getIdNumber, the id
number is : " + idNumber);
            return idNumber;
         }//end getIdNumber
      }
```

```
package olympiad;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;

public class LogoutServlet extends HttpServlet
{
   public void init() throws ServletException
   {}

   protected void doGet(HttpServletRequest request,
HttpServletResponse response)
   throws ServletException, IOException
   {
      HttpSession session = request.getSession();
      session.invalidate();
      String contextPath = request.getContextPath();
      response.sendRedirect(contextPath + "/");
   }
}
```

```
package olympiad;


import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;
import java.util.Enumeration;


public class ModifyArcheryServlet extends HttpServlet
{
   RequestDispatcher modifyArcheryPage;
   RequestDispatcher loginPage;
   RequestDispatcher viewArcheryPage;

   public void init() throws ServletException
   {
        ServletContext context = getServletContext();

        modifyArcheryPage =
context.getRequestDispatcher(Constants.modifyArcheryPagePath);
       if (modifyArcheryPage == null) {
            throw new
ServletException(Constants.modifyArcheryPagePath + " not found");
           }

        loginPage =
context.getRequestDispatcher(Constants.loginPagePath);
       if (loginPage == null) {
            throw new ServletException(Constants.loginPagePath +
" not found");
           }

        viewArcheryPage =
context.getRequestDispatcher(Constants.viewArcheryPagePath);
       if (viewArcheryPage == null) {
            throw new
ServletException(Constants.viewArcheryPagePath + " not found");
           }
   }

    protected void doPost(HttpServletRequest request,
HttpServletResponse response)
       throws ServletException, IOException
      {
       System.out.println("ModifyArcheryServlet running");

       String action = request.getParameter("action");
         if ( action.equals("cancel")) {
           viewArcheryPage.forward(request, response);
           return;
         }

         String id = request.getParameter("id");
         if (id == null || id.equals("")) {
            request.setAttribute("error", "id is missed.");
            modifyArcheryPage.forward(request, response);
```

```
            return;
         }

         String noc = request.getParameter("noc");
         if (noc == null || noc.equals("")) {
            request.setAttribute("error", "noc is missed.");
            modifyArcheryPage.forward(request, response);
            return;
         }

         String event = request.getParameter("event");
         if (event == null || event.equals("")) {
            request.setAttribute("error", "event is missed.");
            modifyArcheryPage.forward(request, response);
            return;
         }

      String familyName = request.getParameter("familyName");
            if (familyName == null || familyName.equals("")) {
            request.setAttribute("error", "The familyName is a
required.");
            modifyArcheryPage.forward(request, response);
          return;
             }

        String givenName = request.getParameter("givenName");
      if (givenName == null || givenName.equals("")) {
            System.out.println("given name is null");
            request.setAttribute("error", "The givenName is a
required.");
            modifyArcheryPage.forward(request, response);
            return;
      }

      String month = request.getParameter("month");
            if (month == null || month.equals("")) {
            request.setAttribute("error", "The month of birthday is
a required.");
            modifyArcheryPage.forward(request, response);
            return;
             }

      String day = request.getParameter("day");
      if (day == null || day.equals("")) {
            request.setAttribute("error", "The day of birthday is a
required.");
            modifyArcheryPage.forward(request, response);
            return;
             }

      String year =  request.getParameter("year");
            if (year == null || year.equals("")) {
            request.setAttribute("error", "The year of birthday is
a required.");
            modifyArcheryPage.forward(request, response);
            return;
             }
```

```
            String gender = request.getParameter("gender");
                  if (gender == null || gender.equals("")) {
                 request.setAttribute("error", "Gender is a required.");
                 modifyArcheryPage.forward(request, response);
                 return;
                  }

            String events[] = request.getParameterValues("events");
            if(events == null || events.length == 0){
                 request.setAttribute("error", "The event(s) is a
required.");
                 modifyArcheryPage.forward(request, response);
                 return;
            }
            else{
               for(int i=0; i<events.length; i++){
                  System.out.println(events[i] + ",");
                  }
            }

            Archery archery = new Archery(id, noc, event,
                  familyName, givenName, month, day, year, gender,
events);

              request.setAttribute("archery", archery);

            try {
              archery.updateDelegate(archery);
            } catch (Exception e) {
                throw new ServletException(e.toString());
                }

            //for getting the new information of delegate after updated
              //Archery archery = null;
              try {
                archery = Archery.findDelegate(id);
              } catch (Exception e) {
                throw new ServletException(e.toString());
                }
              HttpSession session = request.getSession();
              session.setAttribute("archery", archery);

             request.setAttribute("error", "Register sucess, please
register other again.");
              viewArcheryPage.forward(request, response);
         }//end dopost
      }
```

```
package olympiad;


import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;
import java.util.Enumeration;


public class ModifyBasketballServlet extends HttpServlet
{
   RequestDispatcher modifyBasketballPage;
   RequestDispatcher loginPage;
   RequestDispatcher viewBasketballPage;

   public void init() throws ServletException
   {
        ServletContext context = getServletContext();

        modifyBasketballPage =
context.getRequestDispatcher(Constants.modifyBasketballPagePath);
        if (modifyBasketballPage == null) {
             throw new
ServletException(Constants.modifyBasketballPagePath + " not found");
            }

        loginPage =
context.getRequestDispatcher(Constants.loginPagePath);
        if (loginPage == null) {
             throw new ServletException(Constants.loginPagePath +
" not found");
            }

        viewBasketballPage =
context.getRequestDispatcher(Constants.viewBasketballPagePath);
        if (viewBasketballPage == null) {
             throw new
ServletException(Constants.viewBasketballPagePath + " not found");
            }
   }

   protected void doPost(HttpServletRequest request,
HttpServletResponse response)
        throws ServletException, IOException
      {
        System.out.println("ModifyBasketballServlet running");

        String action = request.getParameter("action");
          if ( action.equals("cancel")) {
            viewBasketballPage.forward(request, response);
            return;
          }

          String id = request.getParameter("id");
          if (id == null || id.equals("")) {
             request.setAttribute("error", "id is missed.");
             modifyBasketballPage.forward(request, response);
```

```
                return;
            }

            String noc = request.getParameter("noc");
            if (noc == null || noc.equals("")) {
                request.setAttribute("error", "noc is missed.");
                modifyBasketballPage.forward(request, response);
                return;
            }

            String event = request.getParameter("event");
            if (event == null || event.equals("")) {
                request.setAttribute("error", "event is missed.");
                modifyBasketballPage.forward(request, response);
                return;
            }

        String familyName = request.getParameter("familyName");
              if (familyName == null || familyName.equals("")) {
                request.setAttribute("error", "The familyName is a
required.");
                modifyBasketballPage.forward(request, response);
             return;
                }

          String givenName = request.getParameter("givenName");
        if (givenName == null || givenName.equals("")) {
              System.out.println("given name is null");
              request.setAttribute("error", "The givenName is a
required.");
              modifyBasketballPage.forward(request, response);
              return;
        }

        String month = request.getParameter("month");
              if (month == null || month.equals("")) {
              request.setAttribute("error", "The month of birthday is
a required.");
              modifyBasketballPage.forward(request, response);
              return;
                }

        String day = request.getParameter("day");
        if (day == null || day.equals("")) {
              request.setAttribute("error", "The day of birthday is a
required.");
              modifyBasketballPage.forward(request, response);
              return;
                }

        String year =  request.getParameter("year");
              if (year == null || year.equals("")) {
              request.setAttribute("error", "The year of birthday is
a required.");
              modifyBasketballPage.forward(request, response);
              return;
                }
```

```
            String gender = request.getParameter("gender");
                  if (gender == null || gender.equals("")) {
                  request.setAttribute("error", "Gender is a required.");
                  modifyBasketballPage.forward(request, response);
                  return;
                   }

            String events[] = request.getParameterValues("events");
            if(events == null || events.length == 0){
                  request.setAttribute("error", "The event(s) is a
required.");
                  modifyBasketballPage.forward(request, response);
                  return;
            }
            else{
               for(int i=0; i<events.length; i++){
                  System.out.println(events[i] + ",");
                  }
            }

            Basketball basketball = new Basketball(id, noc, event,
                  familyName, givenName, month, day, year, gender,
events);

              request.setAttribute("basketball", basketball);

            try {
              basketball.updateDelegate(basketball);
            } catch (Exception e) {
                throw new ServletException(e.toString());
                }

            //for getting the new information of delegate after updated
              //Basketball basketball = null;
              try {
                basketball = Basketball.findDelegate(id);
              } catch (Exception e) {
                throw new ServletException(e.toString());
                }
              HttpSession session = request.getSession();
              session.setAttribute("basketball", basketball);

             request.setAttribute("error", "Register sucess, please
register other again.");
              viewBasketballPage.forward(request, response);
         }//end dopost
      }
```

```
package olympiad;


import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;
import java.util.Enumeration;


public class ModifyOfficialServlet extends HttpServlet
{
   RequestDispatcher sorryPage;
   RequestDispatcher viewOfficialPage;

   public void init() throws ServletException
   {
        ServletContext context = getServletContext();

      sorryPage =
context.getRequestDispatcher(Constants.sorryPagePath);
      viewOfficialPage =
context.getRequestDispatcher(Constants.viewOfficialPagePath);

      if (sorryPage == null) {
           throw new ServletException(Constants.sorryPagePath +
" not found");
          }
      if (viewOfficialPage == null) {
           throw new
ServletException(Constants.viewOfficialPagePath + " not found");
          }
   }

   protected void doPost(HttpServletRequest request,
HttpServletResponse response)
      throws ServletException, IOException
     {
      System.out.println("ModifyOfficial servlet running");

      String action = request.getParameter("action");
        if ( action.equals("cancel")) {
          viewOfficialPage.forward(request, response);
          return;
        }

      String id = request.getParameter("id");
            if (id == null || id.equals("")) {
           request.setAttribute("error", "id is missed.");
           sorryPage.forward(request, response);
         return;
      }

      String noc = request.getParameter("noc");
            if (noc == null || noc.equals("")) {
           request.setAttribute("error", "noc is missed.");
           sorryPage.forward(request, response);
         return;
```

```
            }
        String familyName = request.getParameter("familyName");
            if (familyName == null || familyName.equals("")) {
            request.setAttribute("error", "familyName is missed.");
            sorryPage.forward(request, response);
          return;
            }
        System.out.println("familyName is " + familyName);

          String givenName = request.getParameter("givenName");
        if (givenName == null || givenName.equals("")) {
            System.out.println("given name is null");
            request.setAttribute("error", "givenName is missed.");
            sorryPage.forward(request, response);
            return;
        }
        System.out.println("givenName is " + givenName);

        String month = request.getParameter("month");
            if (month == null || month.equals("")) {
            request.setAttribute("error", "The month of birthday is
missed.");
            sorryPage.forward(request, response);
            return;
            }

        String day = request.getParameter("day");
        if (day == null || day.equals("")) {
            request.setAttribute("error", "The day of birthday is
missed.");
            sorryPage.forward(request, response);
            return;
            }

        String year =  request.getParameter("year");
            if (year == null || year.equals("")) {
            request.setAttribute("error", "The year of birthday is
missed.");
            sorryPage.forward(request, response);
            return;
            }

        String gender = request.getParameter("gender");
            if (gender == null || gender.equals("")) {
            request.setAttribute("error", "Gender is missed.");
            sorryPage.forward(request, response);
            return;
            }
        String position = request.getParameter("position");
            if (position == null || position.equals("")) {
            request.setAttribute("error", "The position is
missed.");
            sorryPage.forward(request, response);
            return;
            }

        String event = request.getParameter("event");
```

```
            if(event == null || event.equals("")){
                 request.setAttribute("error", "The event is missed.");
                 sorryPage.forward(request, response);
                 return;
            }

            Official official = new Official(id, noc, position,
                  familyName, givenName, month, day, year, gender,
event);

              request.setAttribute("official", official);

            try {
               official.updateDelegate(official);
            } catch (Exception e) {
                throw new ServletException(e.toString());
                }

            //to get the new information of delegate after update
              //Archery official = null;
              try {
                official = Official.findDelegate(id);
              } catch (Exception e) {
                throw new ServletException(e.toString());
                }
              HttpSession session = request.getSession();
              session.setAttribute("official", official);

            request.setAttribute("error", "You register the official is
success, please register another.");
                 viewOfficialPage.forward(request, response);
          }
      }
```

```
package olympiad;

import java.io.IOException;

public class NocId
{
   public String noc;
   public NocId( String noc){
     this.noc = noc;
     System.out.println( "inside NocId constuctor this.noc is " +
this.noc );
   }

   public String getNocCode()
   {
    String nocCode = "";
    if (noc.equals("usa")) nocCode = "000";
    if (noc.equals("chn")) nocCode = "001";
    if (noc.equals("eng")) nocCode = "002";
    System.out.println( "inside NocId getNocCode noc is " +
nocCode);
    return nocCode;
   }
}
```

```
package olympiad;


import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;

/**
 * Requires users to login successfully in order to access
protected resources.
 */
public class NocUserFilter implements Filter
{
   public void init(FilterConfig filterConfig) throws
ServletException { }
   public void destroy() { }

   /**
    * User is logged in if a user object is stored in session.
    */
   public void doFilter( ServletRequest req,
                         ServletResponse resp,
                         FilterChain chain) throws IOException,
ServletException
   {
      HttpServletRequest request = (HttpServletRequest) req;
      HttpServletResponse response = (HttpServletResponse) resp;
      HttpSession session = request.getSession();
      User user = (User) session.getAttribute("user");
      //Save the requested resource, so we can forward to it after
logging in.
      if (user == null) {
         String requestedResource =
request.getRequestURL().toString();
         session.setAttribute("requestedResource",
requestedResource);
         response.sendRedirect(request.getContextPath() +
Constants.loginPagePath);
         return;
      }
      chain.doFilter(request, response);
   }
}
```

```
package olympiad;


import javax.servlet.http.*;
import javax.naming.*;
import java.sql.*;
import javax.sql.DataSource;
import java.util.Vector;

public class Official
{
   static private DataSource ds;

   static
   {
      try {
         InitialContext ic = new InitialContext();
         Context tomcatContext = (Context)
ic.lookup(Constants.jndiContainerContext);
         ds = (DataSource)
tomcatContext.lookup(Constants.jndiDatabaseName);
         if (ds == null) throw new RuntimeException("no
DataSource");
      } catch (Exception e) {
         throw new RuntimeException(e);
      }
   }

   private String idNumber, nocId, eventId, givenName, familyName,
               month, day, year, gender, position, event;

   /*
    *@ For the official modify and delete use
   */
   public Official( String idNumber, String nocId, String
position, String familyName,
                  String givenName, String month, String day, String
year,
                     String gender, String event )
   {
      this.idNumber = idNumber;
      this.nocId = nocId;
      this.position = position;
            this.familyName = familyName;
      this.givenName = givenName;
            this.month = month;
            this.day = day;
            this.year = year;
      this.gender = gender;
      this.event = event;
   }

   public String getIdNumber() { return idNumber; }
   public String getNocId() { return nocId; }
   public String getEventId() { return eventId; }
   public String getFamilyName() { return familyName; }
   public String getGivenName() { return givenName; }
```

```
        public String getMonth() { return month; }
        public String getDay() { return day; }
        public String getYear() { return year; }
        public String getGender() { return gender; }
        public String getPosition() {return position; }
        public String getEvent() {return event; }

        /*
         * For an individual official, insert the registeration data
once submited.
         */
        public void insertDelegate(Official official) throws Exception
        {
             Connection connection = null;
             try {
                connection = ds.getConnection();
                Statement statement = connection.createStatement();
                String s = "INSERT INTO officials (id, noc, position,
familyName, givenName, " +
                           "month, day, year, gender, event)" +
                           "VALUES ('" + idNumber + "', '" + nocId +
"', '" + position + "', '" +
                           familyName + "', '" + givenName + "', '" +
                           month + "', '" +  day + "', '" +  year + "',
'" + gender + "', '" + event + "' )";
                statement.executeUpdate(s);
             }
             catch(SQLException sqle) {
                  System.out.println("Official class, insertDelegate
method - SQL Exception error:" + sqle);
             }
             catch(Exception e) {
                  System.out.println("Official insertDelegate method -
Exception: " + e);
             }
             finally {
                try{
                     connection.close();
                }catch(SQLException sql){}
             }
         }//end insertDelegate

        public static Official findDelegate(String id) throws Exception
        {
           Official official = null;
           Connection connection = null;
           try {
              connection = ds.getConnection();
              Statement statement = connection.createStatement();
              String s = "select * from officials where id ='" + id +
"';";
              ResultSet rs = statement.executeQuery(s);
              if (!rs.first()) return null;
              String idnull = rs.getString("id");
              String noc = rs.getString("noc");
              String position = rs.getString("position");
              String familyName = rs.getString("familyname");
```

```
                String givenName = rs.getString("givenname");
                String month = rs.getString("month");
                String day = rs.getString("day");
                String year = rs.getString("year");
                String gender = rs.getString("gender");
                String event = rs.getString("event");
                official = new Official(idnull, noc, position,
familyName, givenName,
                                month, day, year, gender, event);
             }
             catch(SQLException sqle) {
                System.out.println("Official class, findDelegate method -
SQL Exception error:" + sqle);
             }
             catch(Exception e) {
                System.out.println("Official class, findDelegate method -
Exception: " + e);
             }
             finally {
                connection.close();
             }
             return official;
          }//end findDelegate

          public static void deleteDelegate (String id) throws Exception
          {
             Connection connection = null;

             try {
               connection = ds.getConnection();
               Statement statement = connection.createStatement();
               //save id for feature adding other delegates in the same
noc
             String s = "insert into idstore (id) values ('" + id +
"');";
               statement.executeUpdate(s);

               s = "delete from officials where id = '" + id + "';";
               statement.executeUpdate(s);
             }
             catch(SQLException sqle) {
               System.out.println("Official class, deleteDelegate method
- SQL Exception error:" + sqle);
             }
             catch(Exception e) {
               System.out.println("Official class, deleteDelegate method
- Exception: " + e);
             }
             finally {
               connection.close();
             }
          }//end deleteDelegate

          public static void updateDelegate(Official delegate) throws
Exception
          {
             String id = delegate.getIdNumber();
```

```
            String position = delegate.getPosition();
            String familyName = delegate.getFamilyName();
            String givenName = delegate.getGivenName();
            String month = delegate.getMonth();
            String day = delegate.getDay();
            String year = delegate.getYear();
            String gender = delegate.getGender();
            String event = delegate.getEvent();

            Connection connection = null;
            try {
              connection = ds.getConnection();
              Statement statement = connection.createStatement();
              String s = "update officials set " +
                         "position = '" + position + "', " +
                           "familyname = '" + familyName + "', " +
                        "givenname = '" + givenName + "', " +
                        "month = '" + month + "', " +
                        "day = '" + day + "', " +
                          "year = '" + year + "', " +
                        "gender = '" + gender + "', " +
                        "event = '" + event + "'" +
                        "where id = '" + id + "'";
              statement.executeUpdate(s);
            }
            catch(SQLException sqle) {
                System.out.println("Official class, updateOfficial
method - SQL Exception error:" + sqle);
              }
              catch(Exception e) {
                System.out.println("Official class, updateOfficial
method - Exception: " + e);
              }
              finally {
                connection.close();
            }
          }//updateDelegate
      }
```

```
package olympiad;


import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;
import java.util.Enumeration;


public class OfficialServlet extends HttpServlet
{
   RequestDispatcher officialPage;
   RequestDispatcher loginPage;
   RequestDispatcher viewListPage;

   public void init() throws ServletException
   {
        ServletContext context = getServletContext();

        officialPage =
context.getRequestDispatcher(Constants.officialPagePath);
        loginPage =
context.getRequestDispatcher(Constants.loginPagePath);
        viewListPage =
context.getRequestDispatcher(Constants.viewListPagePath);

        if (officialPage == null) {
             throw new ServletException(Constants.officialPagePath
+ " not found");
            }
        if (loginPage == null) {
             throw new ServletException(Constants.loginPagePath +
" not found");
            }
        if (viewListPage == null) {
             throw new ServletException(Constants.viewListPagePath
+ " not found");
            }
   }

    protected void doPost(HttpServletRequest request,
HttpServletResponse response)
        throws ServletException, IOException
       {
        System.out.println("Official servlet running");

        String action = request.getParameter("action");
        if ( action.equals("cancel")) {
          viewListPage.forward(request, response);
          return;
        }

        String familyName = request.getParameter("familyName");
              if (familyName == null || familyName.equals("")) {
             request.setAttribute("error", "familyName is a
required.");
             officialPage.forward(request, response);
```

```
                return;
                    }
            System.out.println("familyName is " + familyName);

              String givenName = request.getParameter("givenName");
            if (givenName == null || givenName.equals("")) {
                 System.out.println("given name is null");
                 request.setAttribute("error", "givenName is a
required.");
                 officialPage.forward(request, response);
                 return;
            }
            System.out.println("givenName is " + givenName);

            String month = request.getParameter("month");
                  if (month == null || month.equals("")) {
                 request.setAttribute("error", "The month of birthday is
a required.");
                 officialPage.forward(request, response);
                 return;
                  }

            String day = request.getParameter("day");
            if (day == null || day.equals("")) {
                 request.setAttribute("error", "The day of birthday is a
required.");
                 officialPage.forward(request, response);
                 return;
                  }

            String year =  request.getParameter("year");
                  if (year == null || year.equals("")) {
                 request.setAttribute("error", "The year of birthday is
a required.");
                 officialPage.forward(request, response);
                 return;
                  }

            String gender = request.getParameter("gender");
                  if (gender == null || gender.equals("")) {
                 request.setAttribute("error", "Gender is a required.");
                 officialPage.forward(request, response);
                 return;
                  }

            String position = request.getParameter("position");
                  if (position == null || position.equals("")) {
                 request.setAttribute("error", "The position is a
required.");
                 officialPage.forward(request, response);
                 return;
                  }

            String event = request.getParameter("event");
            if(event == null || event.equals("")){
                 request.setAttribute("error", "The event is a
required.");
```

```
                officialPage.forward(request, response);
                return;
            }

            String idNumber, nocId = "", officialId, nocName = "";
            officialId = EventId.official;

            HttpSession session = request.getSession();
            if (session != null) {
               nocName = (String) session.getAttribute("noc");
               if (nocName == null){
                  request.setAttribute("error", "The session attribute
is null. Please login again");
                  //response.sendRedirect(request.getContextPath() +
Constants.loginPagePath);
                  loginPage.forward (request, response);
                  return;
                }
               NocId nocid = new NocId(nocName);
                 nocId = nocid.getNocCode();
            }

            IdNumber idnumber = new IdNumber(nocId, officialId, gender);

            try {
                idNumber = idnumber.getIdNumber();
            } catch (Exception e) {
                throw new ServletException(e.toString());
                }

            Official official = new Official(idNumber, nocName,
position,
                  familyName, givenName, month, day, year, gender,
event);

              request.setAttribute("official", official);

            try {
              official.insertDelegate(official);
            } catch (Exception e) {
                throw new ServletException(e.toString());
                }
           request.setAttribute("error", "You register the official is
success, please register another.");
            viewListPage.forward(request, response);
         }
      }
```

```
package olympiad;

import java.util.Hashtable;
import javax.naming.*;
import java.sql.*;
import javax.sql.DataSource;
import java.io.IOException;

public class User
{
   static private DataSource ds;

   static
   {
      try {
         InitialContext ic = new InitialContext();
         Context tomcatContext = (Context)
ic.lookup(Constants.jndiContainerContext);
         ds = (DataSource)
tomcatContext.lookup(Constants.jndiDatabaseName);
         if (ds == null) throw new RuntimeException("no
DataSource");
      } catch (Exception e) {
         throw new RuntimeException(e);
      }
   }

   private String userid;
   private String password;
   private String noc;

   public User(String userid, String password, String noc)
   {
      this.userid = userid;
      this.password = password;
      this.noc = noc;
   }

   public String getUserid() { return userid; }
   public String getPassword() { return password; }
   public String getNoc() { return noc; }

   /**
    * @returns null if userid doesn't exist
    */
   public static User find(String userid) throws Exception
   {
      User user = null;
      Connection connection = null;
      try {
         connection = ds.getConnection();
         Statement statement = connection.createStatement();
         String s = "select * from users where userid = '" +
userid + "';";
         ResultSet rs = statement.executeQuery(s);
         if (!rs.first()) return null;
         String password = rs.getString("password");
```

```
               String noc = rs.getString("noc");
              user = new User(userid, password, noc);
            }
            catch(SQLException sqle) {
               System.out.println("User find method - SQL Exception
error:" + sqle);
            }
            catch(Exception e) {
                    System.out.println("User find method - Exception: "
+ e);
            }
            finally {
              try {
                connection.close();
              }catch(SQLException sql){}
            }
            return user;
         }//end User


         public static void update(String userid, String newuserid,
String newpassword) throws Exception
         {
            Connection connection = null;
            try {
               connection = ds.getConnection();
               Statement statement = connection.createStatement();
               String s = "update users set " +
                       "userid = '" + newuserid + "', " +
                           "password = '" + newpassword + "'" +
                       "where userid = '" + userid + "'";
               statement.executeUpdate(s);
            }
            catch(SQLException sqle) {
               System.out.println("User class, update method - SQL
Exception error:" + sqle);
            }
            catch(Exception e) {
                    System.out.println("User class, update method -
Exception: " + e);
            }
            finally {
              try {
                connection.close();
              }catch(SQLException sql){}
            }
         }//end update
      }
```

```
package olympiad;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;

public class ViewArcheryServlet extends HttpServlet
{
   RequestDispatcher modifyArcheryPage;
   RequestDispatcher deleteArcheryPage;
   RequestDispatcher viewArcheryPage;
   RequestDispatcher viewListPage;

   public void init() throws ServletException
   {
      ServletContext context = getServletContext();

      modifyArcheryPage =
context.getRequestDispatcher(Constants.modifyArcheryPagePath);
      if (modifyArcheryPage == null) {
         throw new
ServletException(Constants.modifyArcheryPagePath + " not found");
      }

      deleteArcheryPage =
context.getRequestDispatcher(Constants.deleteArcheryPagePath);
      if ( deleteArcheryPage == null) {
         throw new
ServletException(Constants.deleteArcheryPagePath + " not found");
      }

      viewArcheryPage =
context.getRequestDispatcher(Constants.viewArcheryPagePath);
      if (viewArcheryPage == null) {
         throw new ServletException(Constants.viewArcheryPagePath
+ " not found");
      }

      viewListPage =
context.getRequestDispatcher(Constants.viewListPagePath);
      if (viewListPage == null) {
         throw new ServletException(Constants.viewListPagePath + "
not found");
      }
   }

   protected void doGet(HttpServletRequest request,
HttpServletResponse response)
   throws ServletException, IOException
   {
      System.out.println("ViewArcheryServlet is running");

      String id = request.getParameter("id");
      if (id == null) {
         request.setAttribute("error", "Request error: id value
missing.");
         modifyArcheryPage.forward(request, response);
```

```
            return;
        }

        String action = request.getParameter("action");
        if (action == null) {
            request.setAttribute("error", "Request error: action
value missing.");
            viewArcheryPage.forward(request, response);
            return;
        }

        if (action.equals("cancel")){
          viewListPage.forward(request, response);
          return;
        }

        Archery archery = null;
        try {
            archery = Archery.findDelegate(id);
        } catch (Exception e) {
            throw new ServletException(e.toString());
          }
        HttpSession session = request.getSession();
        session.setAttribute("archery", archery);

        if (action.equals("delete")){
          deleteArcheryPage.forward(request, response);
          return;
        }

        if (action.equals("modify")){
          modifyArcheryPage.forward(request, response);
        return;
        }
    }
}
```

```
package olympiad;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;

public class ViewBasketballServlet extends HttpServlet
{
   RequestDispatcher modifyBasketballPage;
   RequestDispatcher deleteBasketballPage;
   RequestDispatcher viewBasketballPage;
   RequestDispatcher viewListPage;

   public void init() throws ServletException
   {
      ServletContext context = getServletContext();

      modifyBasketballPage =
context.getRequestDispatcher(Constants.modifyBasketballPagePath);
      if (modifyBasketballPage == null) {
         throw new
ServletException(Constants.modifyBasketballPagePath + " not found");
      }

      deleteBasketballPage =
context.getRequestDispatcher(Constants.deleteBasketballPagePath);
      if ( deleteBasketballPage == null) {
         throw new
ServletException(Constants.deleteBasketballPagePath + " not found");
      }

      viewBasketballPage =
context.getRequestDispatcher(Constants.viewBasketballPagePath);
      if (viewBasketballPage == null) {
         throw new
ServletException(Constants.viewBasketballPagePath + " not found");
      }

      viewListPage =
context.getRequestDispatcher(Constants.viewListPagePath);
      if (viewListPage == null) {
         throw new ServletException(Constants.viewListPagePath + "
not found");
      }
   }

   protected void doGet(HttpServletRequest request,
HttpServletResponse response)
   throws ServletException, IOException
   {
      System.out.println("ViewBasketballServlet is running");

      String id = request.getParameter("id");
      if (id == null) {
         request.setAttribute("error", "Request error: id value
missing.");
         modifyBasketballPage.forward(request, response);
```

```
            return;
        }

        String action = request.getParameter("action");
        if (action == null) {
            request.setAttribute("error", "Request error: action
value missing.");
            viewBasketballPage.forward(request, response);
            return;
        }

        if (action.equals("cancel")){
          viewListPage.forward(request, response);
          return;
        }

        Basketball basketball = null;
        try {
            basketball = Basketball.findDelegate(id);
        } catch (Exception e) {
            throw new ServletException(e.toString());
          }
        HttpSession session = request.getSession();
        session.setAttribute("basketball", basketball);

        if (action.equals("delete")){
          deleteBasketballPage.forward(request, response);
          return;
        }

        if (action.equals("modify")){
          modifyBasketballPage.forward(request, response);
        return;
        }
    }
}
```

```
package olympiad;


import javax.servlet.http.*;
import javax.naming.*;
import java.sql.*;
import javax.sql.DataSource;
import java.util.Vector;

public class ViewEvent
{
   static private DataSource ds;

   static
   {
      try {
         InitialContext ic = new InitialContext();
         Context tomcatContext = (Context)
ic.lookup(Constants.jndiContainerContext);
         ds = (DataSource)
tomcatContext.lookup(Constants.jndiDatabaseName);
         if (ds == null) throw new RuntimeException("no
DataSource");
      } catch (Exception e) {
         throw new RuntimeException(e);
      }
   }

   private String idNumber, nocId, eventId, givenName, familyName,
gender,  event;


   /*
    *@ For the noc output delegation list.
   */
   public ViewEvent( String event, String idNumber, String
familyName, String givenName, String gender )
   {
      this.event = event;
      this.idNumber = idNumber;
            this.familyName = familyName;
      this.givenName = givenName;
            this.gender = gender;
   }

   public String getIdNumber() { return idNumber; }
   public String getFamilyName() { return familyName; }
   public String getGivenName() { return givenName; }
   public String getGender() { return gender; }
   public String getEvent() { return event; }
   /**
    * @returns non-null, possibly empty, Vector of all delegates
in noc
    */
   static public Vector findDelegate(String nocId) throws
Exception
   {
```

```
            Vector nocEvent = new Vector();
            Connection connection = null;
            try {
               connection = ds.getConnection();
               Statement statement = connection.createStatement();
               //String s = "select * from athletes where noc = '" +
nocId + "';";
               String s = "select ar.individual, at.id, at.familyname,
at.givenname, at.gender " +
                          "from archery ar, athletes at where at.id =
ar.id and at.noc='" + nocId + "';";
               ResultSet rs = statement.executeQuery(s);
               while (rs.next()) {
                  String event = rs.getString("individual");
                  String id = rs.getString("id");
                  String familyName = rs.getString("familyName");
                 String givenName = rs.getString("givenName");
                  String gender = rs.getString("gender");
                  nocEvent.add(new ViewEvent(event, id, familyName,
givenName, gender));
               }

               s = "select ar.team, at.id, at.familyname, at.givenname,
at.gender " +
                          "from archery ar, athletes at where at.id =
ar.id and at.noc='" + nocId + "';";
               rs = statement.executeQuery(s);
               while (rs.next()) {
                  String event = rs.getString("team");
      System.out.println("view event the tesm is :" + event);
                  String id = rs.getString("id");
                  String familyName = rs.getString("familyName");
                  String givenName = rs.getString("givenName");
                  String gender = rs.getString("gender");
                  nocEvent.add(new ViewEvent(event, id, familyName,
givenName, gender));
               }

              s = "select * from athletes where event = 'bk' and noc='" +
nocId + "';";
               rs = statement.executeQuery(s);
               while (rs.next()) {
                  String event = "bk00";
                  String id = rs.getString("id");
                  System.out.println("viewevent athletes id " + id);
                  String familyName = rs.getString("familyName");
                  String givenName = rs.getString("givenName");
                  String gender = rs.getString("gender");
                  nocEvent.add(new ViewEvent(event, id, familyName,
givenName, gender));
               }
            }
            catch(SQLException sqle) {
               System.out.println("ViewEvent findDelegate method - SQL
Exception error:" + sqle);
            }
            catch(Exception e) {
```

```
                System.out.println("ViewEvent findDelegate method -
Exception: " + e);
             }
             finally {
                connection.close();
             }
             return nocEvent;
          }
       }
```

```
package olympiad;


import javax.servlet.http.*;
import javax.naming.*;
import java.sql.*;
import javax.sql.DataSource;
import java.util.Vector;

public class ViewList
{
   static private DataSource ds;

   static
   {
      try {
         InitialContext ic = new InitialContext();
         Context tomcatContext = (Context)
ic.lookup(Constants.jndiContainerContext);
         ds = (DataSource)
tomcatContext.lookup(Constants.jndiDatabaseName);
         if (ds == null) throw new RuntimeException("no
DataSource");
      } catch (Exception e) {
         throw new RuntimeException(e);
      }
   }

   private String idNumber, nocId, eventId, givenName, familyName,
               month, day, year, gender, position, event;


   /*
    *@ For the noc output delegation list.
   */
   public ViewList( String idNumber, String familyName, String
givenName, String month,
                  String day, String year, String gender, String
event )
   {
      this.idNumber = idNumber;
            this.familyName = familyName;
      this.givenName = givenName;
            this.month = month;
            this.day = day;
            this.year = year;
            this.gender = gender;
      this.event = event;
   }

   public String getIdNumber() { return idNumber; }
   public String getFamilyName() { return familyName; }
   public String getGivenName() { return givenName; }
   public String getMonth() { return month; }
   public String getDay() { return day; }
   public String getYear() { return year; }
   public String getGender() { return gender; }
```

```
        public String getEvent() { return event; }
        /**
         * @returns non-null, possibly empty, Vector of all delegates
in noc
         */
        static public Vector findDelegate(String nocId) throws
Exception
        {
           Vector nocList = new Vector();
           Connection connection = null;
           try {
              connection = ds.getConnection();
              Statement statement = connection.createStatement();
              String s = "select * from officials where noc = '" +
nocId + "';";
              ResultSet rs = statement.executeQuery(s);
              while (rs.next()) {
                 String id = rs.getString("id");
                 String familyName = rs.getString("familyName");
                 String givenName = rs.getString("givenName");
                 String month = rs.getString("month");
                 String day = rs.getString("day");
                 String year = rs.getString("year");
                 String gender = rs.getString("gender");
                 String event = rs.getString("position");
                 nocList.add(new ViewList(id, familyName, givenName,
month, day, year, gender, event));
              }
              s = "select * from athletes where noc = '" + nocId +
"';";
              rs = statement.executeQuery(s);
              while (rs.next()) {
                 String id = rs.getString("id");
                 String familyName = rs.getString("familyName");
                 String givenName = rs.getString("givenName");
               String month = rs.getString("month");
                 String day = rs.getString("day");
                 String year = rs.getString("year");
                 String gender = rs.getString("gender");
                 String event = rs.getString("event");
                 nocList.add(new ViewList(id, familyName, givenName,
month, day, year, gender, event));
                }
           }
           catch(SQLException sqle) {
              System.out.println("ViewList findDelegate method - SQL
Exception error:" + sqle);
           }
           catch(Exception e) {
              System.out.println("ViewList findDelegate method -
Exception: " + e);
           }
           finally {
              connection.close();
           }
           return nocList;
        }
```

```
package olympiad;


import javax.swing.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;

public class ViewListServlet extends HttpServlet
{
   RequestDispatcher sorryPage;
   RequestDispatcher viewOfficialPage;
   RequestDispatcher viewArcheryPage;
   RequestDispatcher viewBasketballPage;

   public void init() throws ServletException
   {
      ServletContext context = getServletContext();

      sorryPage =
context.getRequestDispatcher(Constants.sorryPagePath);
      if (sorryPage == null) {
         throw new ServletException(Constants.sorryPagePath + "
not found");
      }

      viewOfficialPage =
context.getRequestDispatcher(Constants.viewOfficialPagePath);
      if (viewOfficialPage == null) {
         throw new ServletException(Constants.viewOfficialPagePath
+ " not found");
      }

      viewArcheryPage =
context.getRequestDispatcher(Constants.viewArcheryPagePath);
      if (viewArcheryPage == null) {
         throw new ServletException(Constants.viewArcheryPagePath
+ " not found");
      }

      viewBasketballPage =
context.getRequestDispatcher(Constants.viewBasketballPagePath);
      if (viewBasketballPage == null) {
         throw new
ServletException(Constants.viewBasketballPagePath + " not found");
      }

   }

   protected void doGet(HttpServletRequest request,
HttpServletResponse response)
   throws ServletException, IOException
   {
      String id = request.getParameter("id");
      if (id == null) {
         request.setAttribute("error", "Request error: id value
missing.");
```

```
    sorryPage.forward(request, response);
    return;
}

//test the which event delegate page should forward
String event = id.substring(3, 5);

//for view offical delegate record
if (event.equals("30")) {
Official official = null;
try {
  official = Official.findDelegate(id);
} catch (Exception e) {
  throw new ServletException(e.toString());
}

if (official == null) {
    request.setAttribute("error", "Unauthorized access.");
    sorryPage.forward(request, response);
    return;
  }

  HttpSession session = request.getSession();
  if (session != null){
    session.setAttribute("official", official);
    viewOfficialPage.forward(request, response);
  }
}

//for view offical delegate record
if (event.equals("01")) {
  Archery archery = null;
  try {
    archery = Archery.findDelegate(id);
  } catch (Exception e) {
    throw new ServletException(e.toString());
  }

  if (archery == null) {
    request.setAttribute("error", "Unauthorized access.");
    sorryPage.forward(request, response);
    return;
  }

  HttpSession session = request.getSession();
  if (session != null){
    session.setAttribute("archery", archery);
    viewArcheryPage.forward(request, response);
  }
}

//for view offical delegate record
if (event.equals("05")) {
  Basketball basketball = null;
  try {
    basketball = Basketball.findDelegate(id);
  } catch (Exception e) {
```

```
            throw new ServletException(e.toString());
          }

          if (basketball == null) {
            request.setAttribute("error", "Unauthorized access.");
            sorryPage.forward(request, response);
            return;
          }

          HttpSession session = request.getSession();
          if (session != null){
            session.setAttribute("basketball", basketball);
            viewBasketballPage.forward(request, response);
          }
        }
      }//end doGent method
}
```

```
package olympiad;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;

public class ViewOfficialServlet extends HttpServlet
{
   RequestDispatcher modifyOfficialPage;
   RequestDispatcher deleteOfficialPage;
   RequestDispatcher viewListPage;
   RequestDispatcher viewOfficialPage;

   public void init() throws ServletException
   {
      ServletContext context = getServletContext();

      modifyOfficialPage =
context.getRequestDispatcher(Constants.modifyOfficialPagePath);
      if (modifyOfficialPage == null) {
         throw new
ServletException(Constants.modifyOfficialPagePath + " not found");
      }

      deleteOfficialPage =
context.getRequestDispatcher(Constants.deleteOfficialPagePath);
      if ( deleteOfficialPage == null) {
         throw new
ServletException(Constants.deleteOfficialPagePath + " not found");
      }

      viewListPage =
context.getRequestDispatcher(Constants.viewListPagePath);
      if (viewListPage == null) {
            throw new ServletException(Constants.viewListPagePath
+ " not found");
      }

      viewOfficialPage =
context.getRequestDispatcher(Constants.viewOfficialPagePath);
      if (viewOfficialPage == null) {
            throw new
ServletException(Constants.viewOfficialPagePath + " not found");
      }
   }

   protected void doGet(HttpServletRequest request,
HttpServletResponse response)
         throws ServletException, IOException
   {
      System.out.println("ViewOfficialServlet is running");

      String id = request.getParameter("id");
      if (id == null) {
         request.setAttribute("error", "Request error: id value
missing.");
         modifyOfficialPage.forward(request, response);
```

```
            return;
        }

        String action = request.getParameter("action");
        if (action == null) {
            request.setAttribute("error", "Request error: action
value missing.");
            viewOfficialPage.forward(request, response);
            return;
        }

        Official official = null;
        try {
          official = Official.findDelegate(id);
        } catch (Exception e) {
          throw new ServletException(e.toString());
          }
        HttpSession session = request.getSession();
        session.setAttribute("official", official);

        if (action.equals("delete")){
          deleteOfficialPage.forward(request, response);
          return;
        }

        if (action.equals("modify")){
          modifyOfficialPage.forward(request, response);
        return;
        }

        if (action.equals("cancel")){
          viewListPage.forward(request, response);
          return;
        }
      }
    }
```

REFERENCES

1. Martin Fowler with Kendall Scott. UML Distilled - A brief guide to the standard object modeling language. Addison Wesley Longman, July 2001.

2. IOC, Olympic Charter, in force as from 29 November 2002. <http://multimedia.olympic.org/pdf/en report 122.pdf>

3. Hans Bergsten, JavaServer Pages, Third Edition, O"REILLY & Associates, Inc., 2002.

4. Eric Freeman. JavaSpaces Principles, Patterns, and Practice. Addison Wesley, Novermber 1999.

5. Ken Arnold and James Gosling. The Java Programming Language Second Edition. Addison Wesley, February 2000.

6. David M. Geary. Advanced JavaServer Pages. Prentice Hall PTR, 2001.

7. Xue Bai, JavaServer Pages, Thomson Course Technology, 2001.

8. Elmasri and Navathe, Fundamentals of Database Systems, Fourth Edition, Addison Weslsey Longman, Inc., 2003.

9. Raghu Parmakrishna and Johannes Gehrke, Database Management Syatems, Third Edition, McGraw-Hill Higher Education, 2003.

10. IEEE, Recommended Practice for Software Requirements Specifications, IEEE Std 830-1993.

11. PostgreSQL Reference Manual for version 7.3. <http://www.postgresql.org/docs/ >