



SWIM-Suite

A Swim Meet and Swimmer Management Software

Architecture and Design Report

Submitted on: October 25th, 2010

Team Members (in alphabetical order):

AbhishekBindiganavile

Robert Greenberg

SedatOzer

Jay Takle

Contents

1 Introduction	3
1.1 Purpose of the System	3
1.2 Design Goals	3
1.2.1 Performance Criteria	3
1.2.2 Dependability Criteria	4
1.2.3 Cost Criteria	4
1.2.4 Maintenance Criteria	5
1.2.5 End User Criteria	5
1.3 Definitions, Acronyms and Abbreviations	5
1.4 References	6
1.5 System Overview	6
2 Current Software Architecture	7
3 Proposed Software Architecture	8
3.1 Overview	8
3.2 System Decomposition	9
3.3 Hardware-Software Mapping	10
3.4 Persistent Data Management	11
3.4.1 Identification of Persistent Data	11
3.3.2 Storage Management Strategy	11
3.5 Access Control and Security	12
3.6 Global Software Control Flow Design	12
3.7 Boundary Conditions	12
3.7.1 Initialization	12
3.7.2 Termination	12
3.7.3 Exception Handling	13
4 Subsystem Services	14

1. Introduction

1.1 Purpose of the system

Organizing a swim meet is a time consuming job requiring good managerial skills. The organizers have to pay attention to every detail of accepting entries from each swimmer, adding swimmers to the races for which they have registered, seeding them according to their timing and entering each race's results. Special care has to be taken to make sure that the process of accepting swimmer entries and publishing results is done without errors.

The most traditional way used by organizers to process this information is by writing on paper forms. Organizers provide swimmers with paper forms where they have to enter information such as name, age, race, timing, *etc.* Swimmers fill their details and submit their entries to the organizers. Organizers then refer to these forms and put swimmers under the mentioned race. Seeding is done manually. All of this information is stored on sheets of papers.

Software professionals have worked with organizers to develop software that helps the organizers enter, process and store the swimmer information in a more secure, fast, and convenient way. There is software, such as Meet Manager, that envelope features essential for handling each swimmer's data. There are some additional features that organizers feel would be helpful, but are left out in the current software.

Accepting the current software shortcomings as a challenge, we propose a system that would make the swimmer registration process more flexible, integrate it with the swim meet database and provide swimmers with a tool to assess their progress.

1.2 Design Goals

1.2.1 Performance Criteria

Design Criteria	Definition
Response Time	The application must be able to handle requests from the user and return results in a time frame that would not result in a coherent time gap between the request and the result.
Throughput	At any given time frame, only one system functionality can be active in the system as the retrieval of data from the database is in real time.
Memory	A minimum of 20MB is required to run the application

1.2.2 Dependability Criteria

Design Criteria	Definition
Robustness	Swim-Suite must be able to handle a large number of users and to store their details in the database, and the application should have the capability to handle exceptions like invalid inputs without causing the application to crash.
Reliability	Swim-Suite must be able to retrieve and analyze the results of individual swimmers and the information provided to the user must be reasonably accurate.
Availability	Upon receiving a request, the time taken for the result for the request will be the same as the time to retrieve the values from the storage database.
Fault Tolerance	In case of error recognition in the reception of data for a request, the request will be prompted again.
Security	The access control is well-defined and the design is such that authentication is requested where appropriate.
Safety	The system is a web application and uses minimal hardware and would not cause any hazards even in the case of errors and failures.

1.2.3 Cost Criteria

Design Criteria	Definition
Development Costs	Since the system is built using a freely available platform and does not involve licensing fees, the cost of developing the system is significantly less.
Deployment Costs	The application is designed as a web application so installation may be done at minimal cost.

1.2.4 Maintenance Criteria

Design Criteria	Definition
Extensibility	Each layer of the system architecture is implemented as a generic block sothat adding functionality to the blocks is relatively easy.
Modifiability	The functionality of the system can be changed without significant alteration in the overall design and implementation of the system.
Adaptability	The system will possess the capability to be mapped to different application domains.
Portability	The system is currently built as a web application, so it requires a database platform (like MySQL) in order to be hosted in a domain.
Readability	The system is written in a modular form. The blocks are easily understandable for one who has the overview of the system
Traceability of Requirements	The modules of the system will be mapped to the different functionalities of the system.

1.2.5 End User Criteria

Design Criteria	Definition
Utility	The system is highly useful for managing large aquatic sporting events and is useful toboth the organizer and the swimmer.
Usability	The application will be user friendly and visually appealing. The GUI will be simple and informative and data will be presented in an easy-to-understand format.

1.3 Definitions, Acronyms, and Abbreviations

Meet: A meet is a swimming event that consists of many races based on distances and swimming styles.

Race: A race is an individual competition within a meet. Races are mostly based on distances and swimming styles, e.g. 100M Freestyle, etc.

Heat: Depending on the number of swimmers within a particular race, the race is divided into heats. For example, if there 16 swimmers competing in a race and there are only 8 lanes in the swimming pool, then, the race is divided into two heats of 8 swimmers each and the top 4 from each heat compete in a final heat where the winners are decided.

Seed: The swimmers are assigned lanes in the heats depending upon their previous race timings for the same race (“their seeding”).

Organizer: The organizer is the administrator who organizes and manages meets. The organizer uses this application to manage meets.

Swimmer: The swimmer is the end-user who participates in the meets. The swimmer uses the application to register for meets/races, view his/her results, and compare his/her progress over different races.

GUI: Graphical User Interface that allows people to interact with the web application. The GUI offers graphical icons, visual indicators, typed command labels or text navigation to fully represent the information and the actions available to the user.

SDK: Software Development Kit. The SDK is a set of development tools that allows us to create applications for a certain software package.

1.4 References

1. Meet Manager 3.0 – A currently available standalone application to manage swim meets that uses a database to store and retrieve details of meets.

<http://www.hy-tekLtd.com/swim/mm/index.html>

2. FINA is the international governing body of swimming, diving, water polo, synchronized swimming and open water swimming.

<http://www.fina.org>

1.5 System Overview

Swim-Suite will be implemented as an open architecture. Our system is three-tier architecture. The uppermost layer is the portal layer that is directly accessible to the users that will mainly consist of the I/O to the system and the GUI accompanying it. The second layer in the architecture is the application layer. The application layer contains all the generic functions that are called by the portal layer. The application layer communicates directly with the database, which forms the third layer in our system. The database contains all the data that the application handles in the form of tables, and any retrieval or addition to the tables is performed by functions in the application later. The complete system consists of a web application and storage being implemented on a server as a MySQL database. The application will communicate with the server.

The application is targeted as two types of users: Swimmers and Organizers. The data presented will be dependent on the type of user and will be in a format that is suitable to the type of user.

2. Current Software Architecture

The swimming meet manager applications in today's market, especially those used by university recreation facilities, are generally standalone, proprietary, and expensive software. The freely available versions of the software have less functionality than the full versions and impose severe limitations on the user (organizer).

An example of such a system is the aforementioned Meet Manager, which has progressed to version 3.0 over the years. This system stores the swim meet data as a Microsoft Access database. This system is currently priced at around \$250. This application is available as a limited demonstration version for free. However, it contains some serious functionality limitations, and it cannot be used to organize large-scale meets. Also, the current software does not possess the ability to compare progresses of swimmers nor plot race times over various races and meets.

For these reasons, there exists a definite need to develop an application that is easier to access, free-to-use, and includes more functionalities like comparison of race times and ability to view progress plots. *Swim-Suite is that application.*

3. Proposed Software Architecture

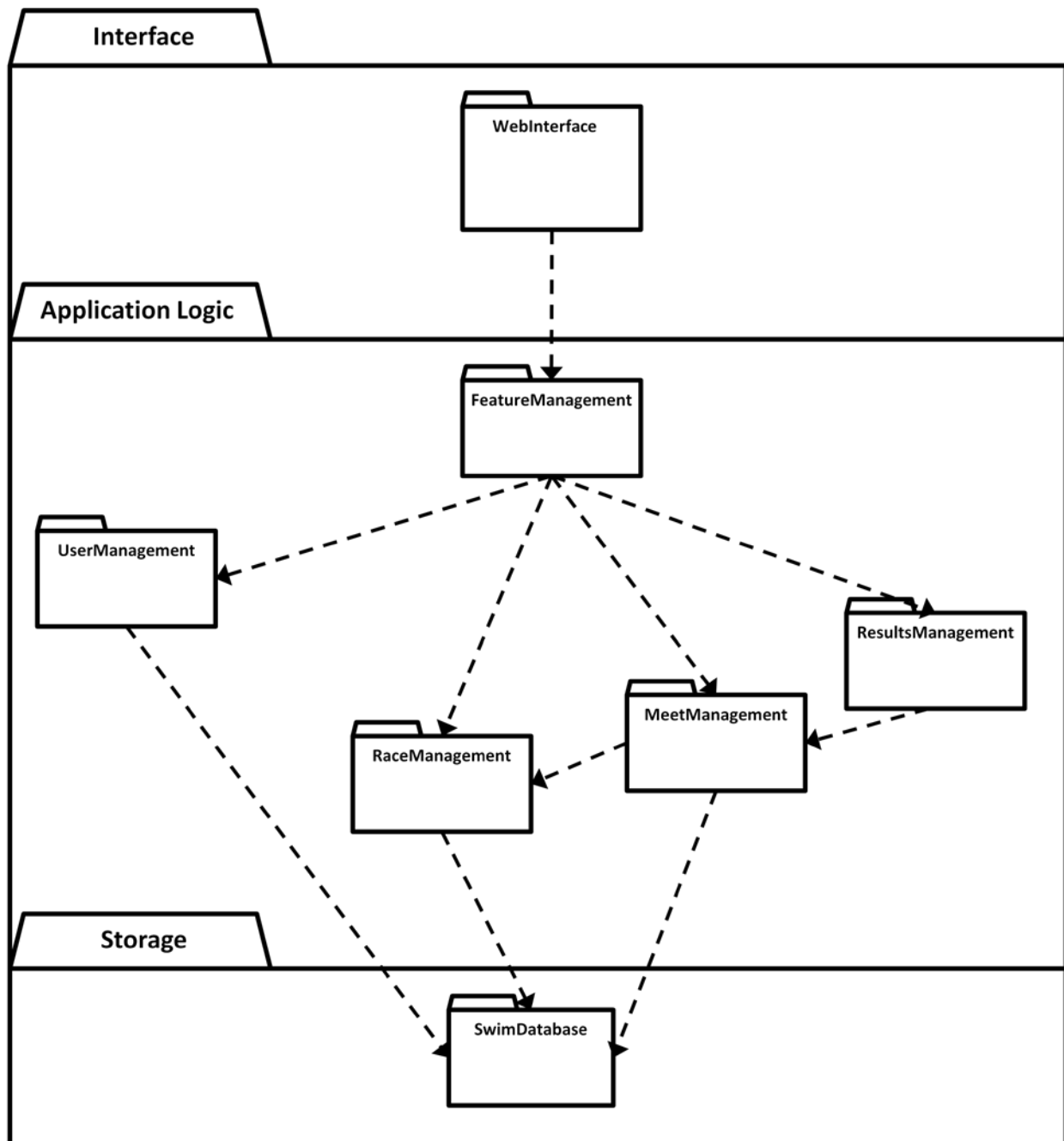
3.1 Overview

The application Swim-Suite will be implemented as an open architecture. Our system will be developed using the **three-tier architecture style**. The uppermost layer is the portal layer that is directly accessible to the users and would primarily consist of the I/O to the system and the GUI accompanying it. The second layer in the architecture is the application layer. The application layer contains all the generic functions that are called by the portal layer. The application layer communicates directly with the database, which forms the third layer in our system. The database contains all the data that the application handles in the form of tables, and any retrieval or addition to the tables is performed by the functions in the application layer. The complete system consists of a web application and storage being implemented on a server as a MySQL database. The application layer will communicate with the server and perform the retrieval and addition of data into the tables.

As mentioned above, layering and partitioning have been applied to the system to obtain a subsystem decomposition of the system. Swim-Suite is decomposed into **7 subsystems** primarily. The only hardware nodes used are desktops or laptop machines for the users. The system does not serve mobile devices like mobile phones or other portable devices.

The application is targeted as two types of users: Swimmers and Organizers. The data that will be presented will be dependent on the type of user and will be in a format that is suitable to the type of user. Authentication methods will be used where appropriate and the content will be delivered depending on the type of user requesting the content.

3.2 Subsystem Decomposition



Subsystem Description

WebInterface: This subsystem provides an interface between the user and other subsystems in the application logic layer of the architecture. It contains the classes AddressFields, Display and Button. AddressFields class provides a text field where the user can type in the destination address. Buttons class is the base class, which is inherited by classes that provide separate functionality when clicked. The Display class provides a display for the webpage that is seen by the user.

FeatureManagement: This subsystem provides an interface between the WebInterface subsystem and the other subsystems in the application logic layer. It is responsible for access control and handles the delegation of control to the other subsystems.

UserManagement: This subsystem handles the different types of users that this application is targeted towards. The UserManagement handles the different functions that need to be performed pertaining to different users. This subsystem communicates with the server directly and retrieves the data as requested by the corresponding user type.

ResultsManagement: This subsystem manages the control of retrieving the results of the requested profile. The ResultsManagement subsystem communicates with the server directly and queries the requested results.

RaceManagement: This subsystem manages the control of a race. A meet will have many races and there needs to be a single-point control for the races. The function of the RaceManagement subsystem is to be that single-point control. The RaceManagement subsystem also handles the control and computation of the number of heats involved in the race. This subsystem calculates the number of heats based on the number of swimmers registered for a particular race.

MeetManagement: This subsystem handles the control of a meet. A meet is created by the organizer. The organizer holds the authorization to create, edit or delete meet. This subsystem is used to handle the various functions performed on the meet.

Server: The server subsystem is basically a set of tables or a database that contains all the information to be handled by the system, from users, races, meets, and results. The server subsystem belongs to the storage layer in the architecture and communicates only with the application logic layer of the system.

3.3 Hardware-Software Mapping

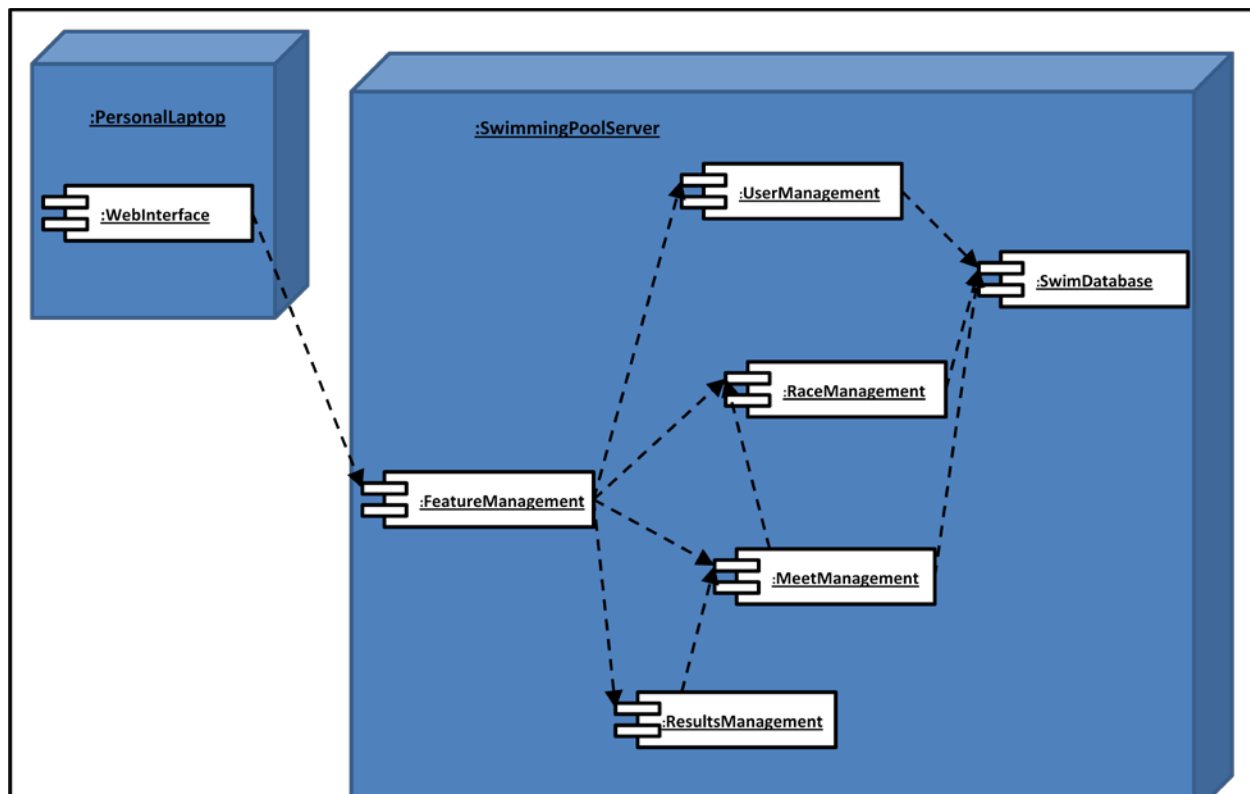
The hardware requirements for the system are as follows:

- Microsoft Windows XP or Vista operating system with at least 20 MB of free space available.
- Single Processor
- Not Distributed
- Any Web browser

In order to host the web application on a local machine, the machine needs to have the MySQL database installed and the Apache Tomcat Application Server installed. All this software is open-source software and is readily available.

Since the system is not distributed all these individual subsystem components are being run on the same system and GUI layer, *i.e.*, the user interface layer is the way that they are all linked together, via calling various functions in the application logic layer to allow access to the features of the other subsystems.

The application may also be hosted on a remote server and the application logic would communicate with the server using Internet Protocol.



3.4 Persistent Data Management

Persistent data is data that must exist in memory longer than a single execution of the software and is retrievable in a later instance of the software. Persistent data must survive both a controlled shutdown and a crash. Persistent data include: swimmer information and meet and race results.

3.4.1 Identification of Persistent Data

In the system, the server is persistent data. The server comprises the primary database for the system. All calls for information regarding the swimmers are sent here and race results are appended to the database.

The database remains in the server after each execution and may be referred upon later executions.

3.4.2 Storage Management Strategy

Storage management strategy affects the overall performance of the system. It is dependent on factors such as: the size and type of data, information density, multiple access, etc.

In the system, the database maintains an extensive data list of various objects that may become rather large in size, such as the Swimmer and Meet objects. Furthermore, multiple users must have access to the database concurrently. There may be any number of swimmers accessing the database at any given time. Prior to the start of the meet, many swimmers will likely be inputting their personal information into the database at approximately the same time.

Based on the specifications and requirements determined for Swim-Suite, the best system has been determined to be a database such as MySQL. Utilizing a MySQL database, multiple online users may be able to access the system simultaneously and efficiently.

3.5 Access Control and Security

Authentication for Swim-Suite will be performed for each actor as necessary. The organizer has full access to all information. There will be far fewer organizers than swimmers. Each meet may have only one organizer but any number of swimmers. Each swimmer will have access to his or her information but will not have access to the information of other swimmers or the organizer(s).

At the execution of the software, prior to the beginning of the meet, each actor must be able to be authenticated to the system. There are login and logout procedures for each user and access will be granted according to the permissions granted to the user.

An example permission matrix:

User	OrganizerApplication	SwimmerApplication	MeetApplication	RaceApplication	Results Application
Organizer	createOrganizerProfile() modifyOrganizerProfile() deleteOrganizerProfile()		createMeet() modifyMeet() listMeet() deleteMeet()	createRace() modifyRace() viewRace() deleteRace()	createResults() storeResults() viewResults()
Swimmer		createSwimmerProfile() modifySwimmerProfile() deleteSwimmerProfile()	listMeet()	viewRace() raceRegister()	viewResults()

3.6 Global Software Control Flow Design

The system is designed with a procedure-driven control. The reason the system is procedure-driven is because all processes in the system are based on input from either the organizers or the swimmers. As the system runs a meet, the organizer inputs data on swimmers and inputs race results. It is only in response to this data input that the system operates.

Specific control flows are described in the state chart diagrams. (See, *Object Oriented Analysis Report*, at pp. 9-10).

In the general case, the system awaits input from the swimmers as to their relevant information, and then the system organizes the seeding of races and heats in a meet. Upon receiving data regarding the races and heats, the system then computes results and designates new seeds, races, and heats accordingly. Finally, the results are inputted and results are available to the users. Then the swimmers request results and specific results are outputted to the specific swimmer.

3.7 Boundary Conditions

Boundary conditions comprise: initialization, termination and exceptions handling.

3.7.1. Initialization:

The system is initialized when the organizer starts the system. This is described in the CreateMeet use case. (See, Sequence Diagrams, *Object Oriented Analysis Report*, at p. 6)

3.7.2 Termination:

The system has no necessary termination point. Swim-Suite is designed to maintain data across several meets. In a typical use case, swimmers may access their data for some time after a meet is concluded. The system may be terminated by the organizer at any time and likely will be terminated at some set time after the conclusion of a swim meet.

Because the data is kept and updated over time, it is necessary for the data to survive application termination. The database on the server should NOT be erased upon termination of the system.

3.7.3. Exceptions Handling:

1) Network Failure: If the data connection is terminated before data is transmitted to or received from the server, the application will need to wait for the connection to be reestablished and the user may then re-transmit or re-request the data.

2) Server Failure: Because of the chosen architecture, the software will not work. All persistent data is stored on the server, and without server functionality, persistent data cannot be accessed.

4. Subsystem Services

4.1 Subsystem webinterface

Subsystem webinterface provides an interface between the user and other subsystems in the application logic layer.

Class Summary	
FormField	This class consists of a text field provided to the user to input his/her text.
Buttons	This class is a base class which describes the template for all the buttons.
Display	This class handles the display functions required to display contents on the screen.

Subsystem webinterface Description

Subsystem webinterface provides an interface between the user and the other subsystems in the lower (application logic layer). It contains the classes FormFields, Buttons and Display. FormFields class provides a text field where the user can input his/her profile details. Buttons class is the base class which performs the respective functionality when the corresponding button is clicked. Display class provides a UI for displaying content as requested on the webpage.

webinterface

Class FormField

`java.lang.Object`

webinterface.FormField

```
public class FormField
extends java.lang.Object
```

This class consists of a text field for the user to input his/her profile details. The text field takes an input of type String. The input from the user is the parameter sent to the webinterface subsystem. It acts as the interface between the user and the application.

Version: 1.0

Constructor Summary	
FormField()	

Method Summary	
void	createForm() Generates the input text field that is displayed when the user starts the application

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail**FormField**

Public **FormField()**

Method Detail**createForm**

public void **createForm()**

throws IOException

Generates the input text field when the user starts the application

Parameters:

form_name - a String storing the name of the user

form_password - a String storing the password of the user

Throws:

IOException - if the input format is incorrect

webinterface**Class Buttons**

java.lang.Object

webinterface.Buttons

public class Buttons

extends java.lang.Object

This class belongs to the base class which describes the template for all the buttons. It is inherited by the subclass buttons each of which provide a different service to the user. It belongs to the input terminal subsystem which is the interface between the user and the application

Version: 1.0

Constructor Summary

Buttons()

Method Summary

void

submit()

This function in the superclass Buttons calls the FeatureManagement subsystem

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail**Buttons**

Public **Buttons**()

Method Detail**submit**

public void **submit**()

This function in the superclass Buttons calls the webinterface subsystem

webinterface**Class Display**

java.lang.Object

View

webinterface.Display

public class Display

extends Display

This class handles the display functions required to display the contents on the screen. This class extends the base class Display.

Version: 1.0

Constructor Summary

Display()

Method Summary

void

displayPage()

Displays the page requested by the user.

Constructor Detail**Display**

```
Public Display()
```

Method Detail**displayPage**

```
public void displayPage()
```

Displays the page requested by the user.

4.2 Subsystem featuremanagement

Subsystem FeatureManagement provides an interface between the webinterface subsystem and other subsystems in the Application Logic layer.

Class Summary

Application	This class manages the content and decision logic to delegate control to the other subsystems.
-------------	--

Subsystem featuremanagement Description

Subsystem FeatureManagement provides an interface between the user and the other subsystems in the lower (application logic layer). It is responsible for access control and delegation of control to the other subsystems. It contains a class Application which takes the user type as an input parameter and activates other subsystems depending on the functionality

featuremanagement

Class Application

```
java.lang.Object
```

```
featuremanagement.Application
```

```
public class Application
```

```
extends java.lang.Object
```

This class manages the control processes in the Application Management subsystem. It is called by the ActionListener functions implemented by the buttons which pass the user inputs to this class. The class triggers the correct control subsystem depending on this input.

Version: 1.0

Constructor Summary

Application()

Method Summary

void	selectUserManagement() It contains the decision logic to be executed for delegation of control to the UserManagement subsystem
void	selectMeetManagement() It contains the decision logic to be executed for delegation of control to the MeetManagement subsystem
void	selectRaceManagement() It contains the decision logic to be executed for delegation of control to the RaceManagement subsystem
void	selectResultsManagement() It contains the decision logic to be executed for delegation of control to the ResultsManagement subsystem

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail**Application**Public **Application()****Method Detail****selectUserManagement**public void **selectUserManagement()**

This function contains the decision logic to be executed for delegation of control to the UserManagement subsystem

Parameters:

type - an object of class Button

selectMeetManagementpublic void **selectMeetManagement()**

This function contains the decision logic to be executed for delegation of control to the MeetManagement subsystem

Parameters:

type - an object of class Button

selectRaceManagement

```
public void selectRaceManagement( )
```

This function contains the decision logic to be executed for delegation of control to the RaceManagement subsystem

Parameters:

type - an object of class Button

selectResultsManagement

```
public void selectResultsManagement( )
```

This function contains the decision logic to be executed for delegation of control to the ResultsManagement subsystem

Parameters:

type - an object of class Button

4.3 Subsystem usermanagement

Subsystem usermanagement handles the control in the application logic layer related to different functionalities of the system pertaining to different users

Class Summary	
OrganizerApplication	This class manages the different functions and the decision logic to delegate control to the different subsystems pertaining to the organizer.
SwimmerApplication	This class manages the different functions and the decision logic to delegate control to the different subsystems pertaining to the swimmer.

Subsystem usermanagement Description

Subsystem usermanagement handles the control in the application logic layer related to different functionalities of the system pertaining to different users. It consists of classes OrganizerApplication and SwimmerApplication. The OrganizerApplication class handles control to the different functions that the organizer is authorized to perform; i.e the control is delegated to the respective subsystems that the organizer has access to. The SwimmerApplication class handles control to the different functions that the swimmer is allowed to perform, i.e, the control is delegated to the respective subsystems that the swimmer has access to.

UserManagement

Class OrganizerApplication

```
java.lang.Object
```

```
    usermanagement.OrganizerApplication
```

```
public class OrganizerApplication
```

```
extends java.lang.Object
```

This class manages the control handling pertaining to the organizer and transfers control to the respective subsystems depending on the functionality requested by the organizer.

Version: 1.0

Constructor Summary

OrganizerApplication()

Method Summary

void	createOrganizerProfile(string ID) This function stores the form details pertaining to the id that is taken as an input parameter, into the database server.
void	modifyOrganizerProfile(string ID) This function submits a query to the database server to update the record pertaining to the ID of the organizer. The ID parameter is taken as in input to the function
void	deleteOrganizerProfile(string ID) This function submits a query to the database server to delete the record pertaining to the ID of the organizer. The ID parameter is taken as in input to the function

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

OrganizerApplication

Public OrganizerApplication()

Method Detail

createOrganizerProfile

public void createOrganizerProfile(string ID)

This function submits a query to the database server to enter the profile details pertaining to the ID that is input as a parameter, into the database

Parameters:

ID - is a string data type that specifies the identity of the user(Organizer) .

modifyOrganizerProfile

public void modifyOrganizerProfile()

This function submits a query to the database server to update the records pertaining to the ID that is input as a parameter.

Parameters:

ID - is a string data type that specifies the identity of the user(Organizer)

deleteOrganizerProfile

```
public void deleteOrganizerProfile()
```

This function submits a query to the database server to delete the records pertaining to the ID that is input as a parameter.

Parameters:

ID - is a string data type that specifies the identity of the user(Organizer)

usermanagement**Class SwimmerApplication**

```
java.lang.Object
```

```
usermanagement.SwimmerApplication
```

```
public class OrganizerApplication
```

```
extends java.lang.Object
```

This class manages the control handling pertaining to the organizer and transfers control to the respective subsystems depending on the functionality requested by the organizer.

Version: 1.0

Constructor Summary

```
OrganizerApplication()
```

Method Summary

void	createSwimmerProfile(string ID) This function stores the form details pertaining to the id that is taken as an input parameter, into the database server.
void	modifySwimmerProfile(string ID) This function submits a query to the database server to update the record pertaining to the ID of the swimmer. The ID parameter is taken as in input to the function
void	deleteSwimmerProfile(string ID) This function submits a query to the database server to delete the record pertaining to the ID of the swimmer. The ID parameter is taken as in input to the function

Methods inherited from class java.lang.Object

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
```

Constructor Detail**SwimmerApplication**

```
Public SwimmerApplication()
```

Method Detail**createSwimmerProfile**

```
public void createSwimmerProfile(string ID)
```

This function submits a query to the database server to enter the profile details pertaining to the ID that is input as a parameter, into the database

Parameters:

ID - is a string data type that specifies the identity of the user(Swimmer).

modifySwimmerProfile

```
public void modifySwimmerProfile()
```

This function submits a query to the database server to update the records pertaining to the ID that is input as a parameter.

Parameters:

ID - is a string data type that specifies the identity of the user(Swimmer)

deleteSwimmerProfile

```
public void deleteSwimmerProfile()
```

This function submits a query to the database server to delete the records pertaining to the ID that is input as a parameter.

Parameters:

ID - is a string data type that specifies the identity of the user(Swimmer)

4.4 Subsystem meetmanagement

Creates, modifies and deletes a swim meet.

Class Summary	
MeetApplication	This class manages the creation, modification and deletion of a swim meet.

Subsystem meetmanagement Description

This subsystem handles the control of a meet. A meet is created by the organizer. The organizer holds the authorization to create, edit or delete meet. This subsystem is used to handle the various functions performed on the meet.

meetmanagement

Class MeetApplication

```
java.lang.Object
```

```
meetmanagement.MeetApplication
```

```
public class MeetApplication
```

```
extends java.lang.Object
```

This class manages the creation, modification and deletion of a swim meet.

Version: 1.0

Constructor Summary

MeetApplication()

Method Summary

void	createMeet() Creates a meet for the organizer to add races.
void	modifyMeet() Changes the details (parameters) of a meet.
java.lang.String	listMeet() Returns a list of available meets.
void	deleteMeet() Deletes selected meet from database.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

MeetApplication

```
Public MeetApplication()
```

Method Detail

createMeet

```
public void createMeet()
```

throws IOException

Generates a meet according to the parameters entered by the organizer.

Parameters:

meetName - a String storing the name of the meet

meetLocation - a String storing the location of the meet

startDate - an Int storing the starting date on the meet

endDate - an Int storing the last date of the meet

entryDeadline - an Int for storing the deadline for entry submission.

ageUpdate - an Int storing the date which is the benchmark for calculating every swimmers age.

type - a String storing the type of meet.

course - a String storing the type of course.

Throws:

IOException - if the input format is incorrect

modifyMeet**public void modifyMeet()**

throws IOException

Changes the details of a meet according to the parameters entered by the organizer.

Parameters:

meetName - a String storing the name of the meet

meetLocation - a String storing the location of the meet

startDate - an Int storing the starting date on the meet

endDate - an Int storing the last date of the meet

entryDeadline - an Int for storing the deadline for entry submission.

ageUpdate - an Int storing the date which is the benchmark for calculating every swimmers age.

type - a String storing the type of meet.

course - a String storing the type of course.

listMeet**public java.lang.String listMeet()**

throws IOException

Returns a list of available meets.

Returns:A list of all the available meets.

deleteMeet**public void deleteMeet()**

throws IOException

Deletes the selected meet from the database.

Parameters:

meetname - a String storing the name of the meet.

Returns:Meet deletion confirmation message.

4.5 Subsystem racemanagement

Creates, modifies and deletes a swim race. It also seeds a heat according to the swimmers earlier timing.

Class Summary	
RaceApplication	This class manages the creation, modification and deletion of a race.
HeatApplication	This class manages the seeding of a heat.

Subsystem racemanagement Description

This subsystem manages the control of a race. A meet will have many races and there needs to be a single point control for the races. The function of the RaceManagement subsystem is precisely that. The RaceManagement subsystem also handles the control and computation of the number of heats involved in the race. This subsystem calculates the number of heats based on the number of swimmers registered for a particular race..

meetmanagement

Class RaceApplication

java.lang.Object

racetmanagement.RaceApplication

```
public class RaceApplication
```

```
extends java.lang.Object
```

This class manages the creation, modification and deletion of a race in a meet. It also seeds a heat according to the every swimmers earlier timing.

Version: 1.0

Constructor Summary

RaceApplication()

Method Summary

void	createRace() Creates a race for the swimmers to register.
void	modifyRace() Changes the details (parameters) of a race.
java.lang.String	viewRace () Returns a list of races.
void	deleteRace() Deletes selected race from meet
void	createHeat() Distributes swimmers registered for a race, into different heats according to their timings.
void	raceRegister() Adds the swimmer to the selected race.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail**RaceApplication**

```
Public RaceApplication()
```

Method Detail**createRace**

```
public void createRace()
    throws IOException
    Creates a race for the swimmers to register.
```

Parameters:
 raceType - a String storing the type of race.
 numberOfLanes - an Integer storing the number of lanes.
 raceID - an Integer storing the race identification number.

modifyRace

```
public void modifyRace()
    throws IOException
    Changes the details (parameters) of a race as entered by the organizer.
```

Parameters:
 raceType - a String storing the type of race.
 numberOfLanes - an Integer storing the number of lanes.

viewRace

```
public java.lang.String viewRace()
    throws IOException
    Returns a list of races.
```

Parameters:
 raceID - a String storing the ID of the race

Returns:
 A list of all the available races in a meet.

Throws:
 IOException - if the input format is incorrect

deleteRace

```
public void deleteRace()
    throws IOException
    Deletes the selected race from the meet.
```

Parameters:
 raceID - an Integer storing the race identification number

Returns:
 Race deletion confirmation message.

Throws:

IOException - if the input format is incorrect

createHeat

public void **createHeat**()

throws IOException

Distributes swimmers registered for a race, into different heats according to their timings.

Parameters:

heatID – an Integer storing the race identification number

timing – an Integer storing a swimmers timing.

Throws:

IOException - if the input format is incorrect

raceRegister

public void **raceRegister**()

throws IOException

Adds the swimmer to the selected race.

Parameters:

raceID – an Integer storing the race identification number.

id – a String storing the swimmers identification code.

timing – an Integer storing a swimmers timing.

Throws:

IOException - if the input format is incorrect

meetmanagement**Class HeatApplication**

java.lang.Object

racetmanagement.HeatApplication

public class HeatApplication

extends java.lang.Object

This class manages the seeding of a heat.

Version: 1.0

Constructor Summary

HeatApplication()

Method Summary

void

createSeeding()

Allots lane number to each swimmer in a heat.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail**HeatApplication**Public **HeatApplication()****Method Detail****createSeeding**public void **createSeeding()**throws **IOException**

Allots lane number to each swimmer in a heat.

Parameters:

numberOfSwimmers – an Integer storing the number of swimmers.

id – a String storing the swimmers identification code.

timing – an Integer storing a swimmers timing.

numberOfLanes – an Integer storing the number of lanes.

heatID – an Integer storing the heat identification number.

Throws:**IOException** – if the input format is incorrect

4.6 Subsystem resultsmanagement

Creates and lists results for a race.

Class Summary

ResultsApplication

This class manages the creation and retrieval of results for a race.

Subsystem resultsmanagement Description

This subsystem manages the control of retrieving the results of the requested profile. The ResultsManagement subsystem communicates with the server directly and queries the requested results.

resultsmanagement**Class ResultsApplication**

```
java.lang.Object
```

```
resultsmanagement.ResultsApplication
```

```
public class ResultsApplication
```

```
extends java.lang.Object
```

This class manages the creation and retrieval of results for a race.

Version: 1.0

Constructor Summary

```
ResultsApplication()
```

Method Summary

Void	<code>createResults()</code> Creates a results table for every heat.
void	<code>storeResults()</code> Stores the results for a race in the database.
java.lang.String	<code>viewResults()</code> Returns a list of results of a race.
void	<code>deleteResults()</code> Deletes the results of a selected race.

Methods inherited from class java.lang.Object

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
```

Constructor Detail**ResultsApplication**

```
Public ResultsApplication()
```

Method Detail**createResults**

```
public void createResults( )
```

```
    throws IOException
```

Creates a results table for every heat, wherein the organizer can manually fill the new timings.

Parameters:

numberOfSwimmers – an Integer storing the number of swimmers.

id – a String storing the swimmers identification code.

raceID – an Integer storing the race identification number

Throws:

IOException – if the input format is incorrect

storeResults

```
public void StoreResults( )
```

```
    throws IOException
```

Stores the results for a race, in the database.

Parameters:

numberOfSwimmers – an Integer storing the number of swimmers.

id – a String storing the swimmers identification code.

raceID – an Integer storing the race identification number

timing – an Integer storing a swimmers timing.

Throws:

IOException – if the input format is incorrect

viewResults

```
public java.lang.String viewResults( )
```

```
    throws IOException
```

Returns a list of results of a race.

Parameters:

raceID – an Integer storing the race identification number.

Returns:

A list of timings for each swimmer in a race.

Throws:

IOException – if the input format is incorrect

deleteResults

```
public void deleteResults()
    throws IOException
    Deletes the selected race results from the database.
```

Parameters:

raceID – an Integer storing the race identification number.

Returns:

Result deletion confirmation message.

Throws:

`IOException` – if the input format is incorrect

4.7 Subsystem SwimDatabase

Subsystem SwimDatabase handles all the data in the system.

Class Summary	
database	This class provides the interface between the Server/Storage layer and the application logic layer.

Subsystem SwimDatabase Description

Subsystem SwimDatabase provides an interface between the Server/Storage Layer and the Application Logic layer of the system. It contains the class database which maintains a list of all the data present in the system in the form of database tables. It accepts queries from the UserManagement, RaceManagement, MeetManagement and ResultManagement subsystems and returns relevant information to it.

SwimDatabase

Class database

```
java.lang.Object
```

```
SwimDatabase.database
```

```
public class database
    extends java.lang.Object
```

This class provides the interface between the Storage/Server layer and the application logic layer.

Version: 1.0

Constructor Summary
database()

Method Summary	
swimdata	submitRequestToServer(string) This function accepts a string ID as as input parameter, the ID could pertain to swimmer, organizer, meet or race ID. The function returns the requested type depending on the function requesting the database.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail
database public database()

Method Detail
createOrganizerProfile

```
public swimdata submitRequestToServer(string ID)
```

This function accepts a string ID as as input parameter, the ID could pertain to swimmer, organizer, meet or race ID. The function returns the requested type depending on the function requesting the database.

Parameters:

ID - is a string data type that specifies the identity of the requesting entity

Throws:

IOException - if the input format is incorrect