**New Codington Festival Online Project -   
Component Class Detailed Design**

**Project:** New Codington Festival Online Project

**System:** Festival Event Registration System – Release 1

Table of Contents

[1 Component Class Design – FERS Release 1 2](#_Toc323884690)

[2 List of Packages and Classes 2](#_Toc323884691)

[3 Package/Class Summary 3](#_Toc323884692)

[4 Other information 9](#_Toc323884693)

[5 Note 10](#_Toc323884694)

# Component Class Design – FERS Release 1

The intention of this document is to provide the information expected to be captured by a Component/Class Design. It is used to design a single component which contains one or more classes.

For this project, the high level information will be provided in a Javadocs (API documentation in HTML format from Java source code. The HTML format is used to add the convenience of being able to hyperlink related documents together. It’s created by JavaDoc, a documentation generator from Sun Microsystems.)

# List of Packages and Classes

|  |  |
| --- | --- |
| **Packages List** | **Classes List** |
| com.accenture.adf.businesstier.controller  com.accenture.adf.businesstier.dao  com.accenture.adf.businesstier.entity  com.accenture.adf.businesstier.service  com.accenture.adf.exceptions  com.accenture.adf.helper  com.accenture.adf.test |  |

# Package/Class Summary

com.accenture.adf.businesstier.controller

com.accenture.adf.businesstier.dao

com.accenture.adf.businesstier.entity

com.accenture.adf.exceptions

com.accenture.adf.helper

com.accenture.adf.test

com.accenture.adf.businesstier.controller

com.accenture.adf.businesstier.dao

com.accenture.adf.businesstier.entity

com.accenture.adf.exceptions

com.accenture.adf.helper

com.accenture.adf.test

Package com.accenture.adf.businesstier.controller

|  |  |
| --- | --- |
| **Class Summary** | |
| EventController | A controller class for receiving and handling all event related transactions from the User Interface including finding all available events in the Event Catalog. |
| VisitorController | A controller class for receiving and handling all visitor related transactions from the User Interface including visitor account access, visitor account maintenance, and visitor event registration requests. |

Package com.accenture.adf.businesstier.dao

|  |  |
| --- | --- |
| **Class Summary** | |
| EventDAO | A Data Access Object (DAO) class for handling and managing event related data requested, updated, and processed in the application and maintained in the database. The interface between the application and event data persisting in the database.  *Please find the Pseudocode for the incomplete methods below*  **showAllEvents**()   * Create a new connection to the database * Prepare a statement object using the connection to  query database   + **Query Location:** sqlBean   + **Query:** get all the events from the event table * Log the query that is about to be executed * Execute the SQL statement and keep a reference to the result set * Log the number of events that were retrieved * Log some easily recognizable information about each event * Using a loop:   + copy all event records in the query result-set to the ArrayList of Event objects   + This ArrayList of Events will be returned to the calling method * Close the connection to the result set. * Close the connection to database * Return the ArrayList to the calling method. * **NOTE**: As can be seen in the method signature, ClassNotFoundException and SQLException are thrown in this method. These are caught using ‘try/catch’ blocks in the Service classes   **updateEventNominations**()   * Create a new connection to the database * Prepare a statement object using the connection to  query database   + **Query Location:** sqlBean   + **Query:** decrease the seats available for the event by 1 (in the EVENT table). * Log the query that is about to be executed * Execute the SQL statement and keep a reference to the result set. * If no records are updated or a SQL Error occurs throw a FERSGenericException with the message stating “the event record was not updated. Throw an Exception object.” * Log if the update was successful or unsuccessful and other pertinent information * Close the connection to database * **NOTE**: As can be seen in the method signature, ClassNotFoundException, SQLException and Exception are thrown in this method. Also within the method a customized exception FERSGenericException is thrown. These Exceptions are caught using ‘try/catch’ blocks in the Service classes.   **checkEventsofVisitor()**   * Create a new connection to the database * Log to the logfile the pertinent information passed to the method (Visitor ID, Visitor Name, and the event ID) to be used in the query. * Prepare a statement object using the connection to  query database   + **Query Location:** sqlBean   + **Query:** count of the number of times the visitor signed up for an event from EVENTSESSIONSIGNUP table * Execute the SQL statement and keep a reference to the result set. * Get and store the event count from the result set. * Log the number of times the visitor is registered * Close the connection to the result set. * Close the connection to database * Check the count: If the count is 1 or more, return TRUE, else return FALSE. * **NOTE**: As can be seen in the method signature, ClassNotFoundException and SQLException are thrown in this method. These are caught using ‘try/catch’ blocks in the Service classes |
| VisitorDAO | A Data Access Object (DAO) class for handling and managing visitor related data requested, used, and processed in the application and maintained in the database. The interface between the application and visitor data persisting in the database.  *Please find the Pseudocode for the incomplete methods below*  **insertData()**   * Create a connection to database * Create two statement objects from the connection that uses a query that should be retrieved from the sqlBean * Prepare two statement objects using the connection to  query database   + **Query Location:** sqlBean   + **Query -1:** select all the usernames from the visitor table   + **Query -2:** inserts visitor information into the visitor table * Log Query-1 that is about to be executed * Execute the statement object with Query-1 and keep a reference to the result set. * Using a LOOP, check if the username to be added is in the result-set   + If the username **is** **not** in the visitor table     - Set all the needed parameters for Query-2 using the values in the Visitor object     - Log Query-2 that is about to be executed     - Execute the statement object with Query-2 and keep a reference to the result set.     - If the insert does not work (return value <= 0), throw a FERSGenericException with the message Visitor <Visitor Name> details not inserted into database     - Log the Visitor details were added to the database     - Close the connection to the result set     - Close the connection to the database     - Return TRUE   + If the username **is** in the database     - Log that the username is already being used in the database. * Close the connection to the result set * Close the connection to the database * Return FALSE * **NOTE**: As can be seen in the method signature, ClassNotFoundException, SQLException and Exception are thrown in this method. Also within the method a customized exception FERSGenericException is thrown. These Exceptions are caught using ‘try/catch’ blocks in the Service classes.   **searchUser**()   * Create a connection to database * Prepare a statement object using the connection to  query database   + **Query Location:** sqlBean   + **Query:** retrieve all the data from the visitor table based on the username and password provided. * Log the query that is about to be executed * Execute the query and store the results in a result set * Log that the visitor details were searched using the username and the number of records that were retrieved. * Using a WHILE LOOP, store the record in the resultset in the visitor object * Close the connection to the result set * Close the connection to the database * Return the visitor object * **NOTE**: As can be seen in the method signature, ClassNotFoundException and SQLException are thrown in this method. These are caught using ‘try/catch’ blocks in the Service classes   **registerVisitortoEvent**()   * Create a connection to the database * Log the visitor and event that will be registered to the visitor * Prepare a statement object using the connection to  query database   + **Query Location:** sqlBean   + **Query:** inserts the visitor ID and event ID into the EVENTSESSIONSIGNUP table. * Log the query that is about to be executed * Execute the query to perform the update and store the results in a result set * If insert statement fails throw a FERSGenericException with the message * Close the connection to the result set * Close the connection to the database * **NOTE**: As can be seen in the method signature, ClassNotFoundException, SQLException and Exception are thrown in this method. Also within the method a customized exception FERSGenericException is thrown. These Exceptions are caught using ‘try/catch’ blocks in the Service classes.   **registeredEvents**()   * Create a connection to the database * Prepare a statement object using the connection to  query database   + **Query Location:** sqlBean   + **Query:** Retrive all the events that are registered to a visitor * Set the parameter to the SQL Query * Log the query that is about to be executed * Execute the query to retrieve the event information into a result set * Log the number of events or the list of events that are in the result set * Using a loop:   + copy all event records in the query result-set to the ArrayList of Event objects   + This ArrayList of Events will be returned to the calling method * Close the connection to the resultset * Close the connection to the database * Return the reference to the ArrayList of Events * **NOTE**: As can be seen in the method signature, ClassNotFoundException and SQLException are thrown in this method. These are caught using ‘try/catch’ blocks in the Service classes |

Package com.accenture.tdf.businesstier.entity

|  |  |
| --- | --- |
| **Class Summary** | |
| Event | A Plain Old Java Object (POJO) entity class that stores (and helps to persist in the application) some or all information from the Event Table in the database. |
| People | A Plain Old Java Object (POJO) entity class that stores (and helps to persist in the application) some of the information from the Visitor Table. People is the parent class of the Visitor Class. |
| Visitor | A Plain Old Java Object (POJO) entity class that stores (and helps to persist in the application) more information from the Visitor Table. |

Package com.accenture.adf.businesstier.service

|  |  |
| --- | --- |
| **Interface Summary** | |
| EventFacade | An interface for defining and enforcing operations needed for the Event Service class. It provides the scope of possible database requests made through the EventDAO. |
| VisitorFacade | An interface for defining and enforcing operations needed for the Visitor Service Class. It provides the scope of possible database requests made through the VisitorDAO. |

|  |  |
| --- | --- |
| **Class Summary** | |
| EventServiceImpl | A service class that implements the EventFacade.  Makes event-related data requests to the EventDAO class.  *Please find the Pseudocode for the incomplete methods below*  **getAllEvents**()   * Create an EventDAO object * Create an ArrayList collection of Event class type * In the try/catch block   + Load Events from showAllEvents(…) method via the EventDAO object into the ArrayList   + Keep a reference to the returned List of event objects.   + If no exceptions occur, return the reference.   + If ClassNotFoundException and SQLException exceptions occur, catch them and return null.   **checkEventsofVisitor**()   * Create an EventDAO object * Create a Boolean variable * In the try/catch block   + Load checkEventsofVisitor(…) method via the EventDao object into the Boolean  variable   + Keep a *reference* to the returned Boolean value.   + If no exceptions occur, return the *reference*.   + If ClassNotFoundException and SQLException exceptions occur, catch them , log the Exception message and return null.   **updateEventDeletions**()   * Create an EventDAO object * In the try/catch block   + Call updateEventDeletions (…) in a new DAO object.   + If ClassNotFoundException and SQLException exceptions occur, catch them , log the Exception message. |
| VisitorServiceImpl | A service class that implements the VisitorFacade.  Makes visitor-related data requests to the VisitorDAO class.  *Please find the Pseudocode for the incomplete methods below*  **createVisitor**()   * Create a VisitorDAO object * Create a Boolean variable * In the try/catch block   + Call insertData(…) from the DAO object and reference the return Boolean variable   + If no exceptions occur, return the *reference*.   + If ClassNotFoundException, Exception and SQLException exceptions occur, catch them , log the Exception message and return null.   **searchVisitor**()   * Create a VisitorDAO object * Create a Visitor Object * In the try/catch block   + Call searchUser(…,…) from the DAO object and copy result into the Visitor object   + Catch all possible exceptions and log the provided exception message.   + If no exceptions occur, return the Visitor Object *reference*.   + If ClassNotFoundException and SQLException exceptions occur, catch them , log the Exception message and return null.   **registerVisitor**()   * Create a VisitorDAO object * Create an EventDAO Object * In the try/catch block   + Call registerVisitorToEvent (…) of the VisitorDAO class, using the VisitorDAO object   + Call updateEventNominations() of the EventDAO class, using the EventDAO object   + If ClassNotFoundException, Exception and SQLException exceptions occur, catch them , log the Exception message and return null.   **showRegisteredEvents**()   * Create a VisitorDAO Object * Create an ArrayList of Event Objects * In the try/catch block   + Call registeredEvents () from the DAO object and reference the return ArrayList of Event objects   + If no exceptions occur, return the reference to the list of event objects   + If ClassNotFoundException and SQLException exceptions occur, catch them , log the Exception message and return null. |

Package com.accenture.adf.exceptions

|  |  |
| --- | --- |
| **Exception Summary** | |
| FERSGenericException | A customized Exception class that redirects how the ClassNotFoundException, SQLException and Exception objects are handled in the application. |

Package com.accenture.adf.helper

|  |  |
| --- | --- |
| **Class Summary** | |
| FERSDataConnection | A helper class that centralizes the management of data connections in the underlying database. |
| FERSDbQuery | A helper class that makes available to the application SQL queries that are externalized and centrally maintained using the Spring Framework. |

Package com.accenture.adf.test

|  |  |
| --- | --- |
| **Class Summary** | |
| TestEventDAO | A JUnit test case for the EventDAO class. It is used to test all repository methods which call database sub-routines.  Code contains ‘TODO’ statements. Include the ‘test scripts’ in place of TODO statements to complete the code. |
| TestVisitorDAO | A Junit test case for the VisitorDAO class. It is used to test all repository methods which call database sub-routines.  Code contains ‘TODO’ statements. Include the ‘test scripts’ in place of TODO statements to complete the code. |

# Other information

Please find the Logger and ActionContext classes information.

**LOGGER**:

* + Logger API from “org.apache.log4j.Logger” is used for handling all transaction messages in the classes in FERS Application.
  + Please find an example of logger usage below
  + **Import statement:**
    - import org.apache.log4j.Logger;
  + **Declaration**:
    - private static Logger log = Logger.getLogger(EventDAO.class);
  + **Usage within class:**
    - Example 1: log.info("All Events retreived from Database :" + eventList);
    - Example 2: log.info("Exception is :" + exception.getMessage());

**ActionContext**:

* Actioncontext from Spring Framework is used in FERS Application
* Please find example(s) of ActionContext usage below
  + **Import statements:**
    - import org.springframework.context.ApplicationContext;
    - import org.springframework.context.support.ClassPathXmlApplicationContext;
* **Declaration and usage within class’s CONSTRUCTOR:**
* ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
* query = (FERSDbQuery) context.getBean("SqlBean"); //FERSDbQuery is a FERS class. Please see table for details.

# Note

For more examples of templates and completed component/class designs, refer to the Component Class Design document samples and Templates in ADM for Custom Development v 5.1 Controlled Release.