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

**Project:** New Codington Festival Online Project

**System:** Festival Event Registration System – Release 2

Table of Contents

[1 Component Class Design – FERS Release 2 2](#_Toc323961277)

[2 List of Packages and Classes 2](#_Toc323961278)

[3 Package/Class Summary 2](#_Toc323961279)

[4 Other information 8](#_Toc323961280)

[5 Note 9](#_Toc323961281)

# Component Class Design – FERS Release 2

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.tdf.businesstier.controller  com.accenture.tdf.businesstier.dao  com.accenture.tdf.businesstier.entity  com.accenture.tdf.businesstier.service  com.accenture.tdf.exceptions  com.accenture.tdf.helper  com.accenture.tdf.test | Event  EventController  EventCoordinator NEW  EventDAO  *EventFacade*  EventServicelmpl  FERSDataConnection  FERSDbQuery  FERSGenericException  People  TestEventDAO  TestVisitorDAO  Visitor  VisitorController  VisitorDAO  *VisitorFacade*  VisitorServiceImpl |

# Package/Class Summary

Package com.accenture.tdf.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.tdf.businesstier.dao

Note: Algorithms include only the method names. For the complete signature, please refer to the Javadoc provided as reference for the High Level Component/Class Design.

|  |  |
| --- | --- |
| **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.  **NOTE:**In implementation of the algorithms, 8 ArrayList Objects are created to transfer content between the Business tier and Database.  This information includes the 8 columns in the Event table (eventid, name, description, duration,eventtype,places, seatsavailable, eventsessionid). This will be replaced with TransferObjects in the next version of the Sample application code in the ADF 2.0 Java course.  *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 * Create an ArrayList that will hold an Array of Objects * Using a loop:   + If there is another record, create an array of objects of size 8   + Store each record in the result set returned in an object by setting the values of each object in the array to a value retrieved in the Result set   + Add the array of objects to the ArrayList   + 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   **showAllEvents**(String)   * 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 when given part or all of an event name * 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 * Create an ArrayList that will hold an Array of Objects * Using a loop:   + If there is another record, create an array of objects of size 8   + Store each record in the result set returned in an object by setting the values of each object in the array to a value retrieved in the Result set   + Add the array of objects to the ArrayList   + 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   **showAllEventsAsc**()   * Create a new connection to the database * Prepare a statement object using the connection to query database   + **Query Location:** sqlBean   + **Query**: gets all the events from the event table in ascending order by event name * 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 * Create an ArrayList that will hold an Array of Objects * Using a loop:   + If there is another record, create an array of objects of size 8   + Store each record in the result set returned in an object by setting the values of each object in the array to a value retrieved in the Result set   + Add the array of objects to the ArrayList   + 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.   **showAllEventsDesc**()   * Create a new connection to the database * Prepare a statement object using the connection to query database   + **Query Location:** sqlBean   + **Query**: gets all the events from the event table in descending order by event name * 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 * Create an ArrayList that will hold an Array of Objects * Using a loop:   + If there is another record, create an array of objects of size 8   + Store each record in the result set returned in an object by setting the values of each object in the array to a value retrieved in the Result set   + Add the array of objects to the ArrayList   + 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   **getEvent**(int eventId, int sessionId)   * Create a new connection to the database * Prepare a statement object using the connection to query database   + **Query Location:** sqlBean   + **Query**: get event from database by provided event id and session id * Log the query that is about to be executed * Execute the SQL statement and keep a reference to the result set * Log the id of event that is retrieved * Using a loop:   + Create Event object and set the values using setter methods by fetching it from the resultset. * Close the connection to the result set. * Close the connection to database * Return the Event 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   **insertEvent**(Event insertEvent)   * Create a new connection to the database * Prepare a statement object using the connection to query database   + **Query Location:** sqlBean   + **Query**: Insert event from database by provided event object values * Log the query that is about to be executed * Execute the SQL statement and keep a reference to the result set * Log the id of event that is retrieved * Close the connection to the result set. * Close the connection to database * Return the Event 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   **updateEvent**(Event updateEvent)   * Create a new connection to the database * Prepare a statement object using the connection to query database   + **Query Location:** sqlBean   + **Query**: Update event in database by provided event object values * Log the query that is about to be executed * Execute the SQL statement and keep a reference to the result set * Log the id of event that is retrieved * Close the connection to the result set. * Close the connection to database * Return the Event 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   **deleteEvent**(int eventId, int sesssionId)   * Create a new connection to the database * Prepare a statement object using the connection to query database   + **Query Location:** sqlBean   + **Query**: Delete event from database by providing eventId and sessionId * Log the query that is about to be executed * Execute the SQL statement and keep a reference to the result set * Log the id of event that is retrieved * Close the connection to the result set. * Close the connection to database * Return the Event 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   **getEventCoordinator**()   * Create a new connection to the database * Prepare a statement object using the connection to query database   + **Query Location:** sqlBean   + **Query**: gets all the event coordinator names from the eventcoordinator 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 * Create an ArrayList that will hold an Array of Objects * Using a loop:   + If there is another record, create an EventCoordinator object   + Store each record in the result set returned in an object by setting the values of each object in the eventCoordinator object to a value retrieved in the Result set   + Add the eventCoordinator object to an arraylist   + This ArrayList of EventCoordinator 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 |
| 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.  **NOTE:**In implementation of the algorithms, 9 ArrayList Objects are created to transfer content between the Business tier and Database.  This information includes the 8 columns in the Event table (eventid, name, description, duration, eventtype, places, firstname+lastname, eventsessionid and signupid). This will be replaced with TransferObjects in the next version of the Sample application code in the ADF 2.0 Java course.  *Please find the Pseudocode for the incomplete methods below*  **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 * Create an ArrayList that will hold an Array of Objects * Using a loop:   + If there is another record, create an array of objects of size 9   + Store each record in the result set returned in an object by setting the values of each object in the array to a value retrieved in the Result set   + Add the array of objects to the ArrayList   + 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   **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.  **unregisterEvent**()   * 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:** deletes the visitor, event, and session IDs 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 update 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. |

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. |
| EventCoordinator | A Plain Old Java Object (POJO) entity class that stores (and helps to persist in the application) some or all information from the EventCoordinator 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.tdf.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 |
| VisitorServiceImpl | A service class that implements the VisitorFacade. Makes visitor-related data requests to the VisitorDAO class |

Package com.accenture.tdf.exceptions

|  |  |
| --- | --- |
| **Exception Summary** | |
| FERSGenericException | A customized Exception class that redirects how the ClassNotFoundException, SQLException and Exception objects are handled in the application. |
| Overview Package Class Use Tree Deprecated Index Help | |

Package com.accenture.tdf.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.tdf.test

|  |  |
| --- | --- |
| **Class Summary** | |
| TestEventDAO | A JUnit test case for the EventDAO class. It is used to test all repository methods which make call database sub-routines. |
| TestVisitorDAO | A Junit test case for the VisitorDAO class. It is used to test all repository methods which call database sub-routines. |
| TestEventServiceImpl | A Junit test case for the EventServiceImpl class. It is used to test all repository methods which call service sub-routines |
| TestVisitorServiceImpl | A Junit test case for the VisitorServiceImpl class. It is used to test all repository methods which call service sub-routines |
| TestEventController | A Junit test case for the EventController class. It is used to test all repository methods which call controller sub-routines |
| TestVisitorController | A Junit test case for the VisitorController class. It is used to test all repository methods which call controller sub-routines |
| AssemblyTestEventServiceImpl | A Junit test case for the EventServiceImpl class. It is used to test all repository methods which call service sub-routines |
| AssemblyTestVisitorServiceImpl | Assembly test case for the VisitorServiceImpl class. It is used to test all repository methods which call service sub-routines |
| AssemblyTestEventController | Assembly Junit test case for the EventController class. It is used to test all repository methods which call controller sub-routines |
| AssemblyTestVisitorController | Assembly Junit test case for the VisitorController class. It is used to test all repository methods which call controller sub-routines |

# 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.