Understanding Repository - 06

Eurico Costa

CIT-360 W16

# MVC Pattern

Although I did not have time to conclude the MVC controller in my Sockets example, I have implemented the skeleton of the pattern both on the client (incomplete) and the server side (complete). The implementation can be seen in the following classes:

Example of View: <https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2002/Hibernate/MPFinderClient/src/com/cit360/mpfinder/client/view/MainView.java>

Example of Model: <https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2002/Hibernate/MPFinderClient/src/com/cit360/mpfinder/client/model/Person.java>

Example of Controller: <https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2002/Hibernate/MPFinderClient/src/com/cit360/mpfinder/client/controller/Controller.java>

In my client, similarly to what I did for the server, I decided to not hard code the parameters such as port number and host name, and stored them in a properties file. I’ve created a class that reads the properties from a config file and passes them along to the client code:

Properties File: <https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2002/Hibernate/MPFinderClient/resources/mpfinderclient.config.properties>

MPFinderProperties Class: <https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2002/Hibernate/MPFinderClient/src/com/cit360/mpfinder/client/util/MPFinderProperties.java>

# Application Controller Pattern

In the sockets server, I implemented the Application controller pattern, that can be seen in the following classes. The controller controls the flow of the methods being called dependent of the request method being sent to the sockets server:

Application Controller: <https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2002/Hibernate/mpfinder/src/com/cit360/mpfinder/server/controller/ApplicationController.java>

Controller: <https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2002/Hibernate/mpfinder/src/com/cit360/mpfinder/server/controller/Controller.java>

Handlers: <https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2002/Hibernate/mpfinder/src/com/cit360/mpfinder/server/controller/CreateMissingPersonHandler.java>

And

<https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2002/Hibernate/mpfinder/src/com/cit360/mpfinder/server/controller/GetMissingPersonsHandler.java>

HandlerMap: <https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2002/Hibernate/mpfinder/src/com/cit360/mpfinder/server/controller/HandlerMap.java>

Model: <https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2002/Hibernate/mpfinder/src/com/cit360/mpfinder/server/model/Person.java>

# Sockets IO

The sockets IO server is fully functional, uses most of the technology topics learned during this year, and although the client was is not complete, the server can be tested by connecting to the Sockets Server via telnet, and passing the parameters in the following sequence:

1 – Start the Server

2 – Connect to the server via telnet: telnet localhost 4497 (or whatever port is configured in the server properties file

3 – Once the server returns “ready”, type “ready” to state that the client is ready.

4 – Pass a JSON string to the server, which can be one of the following two, as the only methods that were implemented:

**getMissingPersons**: {"request":"getMissingPersons","data":{}}

**createMissingPerson**: {"request":"createMissingPerson","data":{"authorEmail": "ejcosta.byui@gmail.com","notes": [],"entryDate": "08-APR-2016","givenName": "Luisa","sex": "F","fullName": "Luisa Lima","description": "5'8, brown hair, heavy set. Was last seen wearing a green skirt, white blouse, black shoes.","authorPhone": "(801) 915-6813","dateOfBirth": "19-JUN-1968","homeNeighborhood": "Bairro dos Pescadores","homePostalCode": "4850","expiryDate": "08-OCT-2016","sourceDate": "08-APR-2016","alternateNames": "Luisa","homeStreet": "Rua das Sardinhas, 3 Esq","authorName": "Eurico Costa","homeCity": "Povoa do Varzim","familyName": "Lima","sourceName": "MissingPersonFinder.java","homeCountry": "Portugal","age": 47}}

The server main class is found here: <https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2002/Hibernate/mpfinder/src/com/cit360/mpfinder/server/MissingPersonFinderServer.java>

As mentioned earlier, it implements the Application Controller patterns. I have also included other patterns that one would usually implement in a real-life code example such as:

**Properties Files**

Configuration parameters, such as hibernate parameters (database connection, credentials, etc., as well as port number for the server, jdbc driver, etc.) were defined in an application properties file. This way, these parameters are not hard coded, and if required, they can be easily changed in the properties file and picked up by the application after it restarts:  
  
Properties file for the server: <https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2002/Hibernate/mpfinder/resources/mpfinderserver.config.properties>  
  
Singleton Class that handles the properties file: <https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2002/Hibernate/mpfinder/src/com/cit360/mpfinder/server/util/MPFinderProperties.java>  
  
The code that reads the properties and uses them, in the code is the following (found in the main class of the server application):  
  
**int** portNumber = 0;

//A port number is obtained from properties file "mpfinderserver.config.properties"

//Class MPFinderProperties is the class onto which the properties file is loaded using a //singleton pattern.

**try**{

portNumber = Integer.*parseInt*(MPFinderProperties.*getProperties*().getProperty("server.port"));

}

**catch**(NumberFormatException e){

String err = "Exception: A port number was not supplied or there was a problem obtaining one.”;

System.***out***.println(err);

MissingPersonFinderServer.*getLogger*().error(err);

System.*exit*(99);

}

**Logging**

The server application implements the Log4J framework for logging of application events. The Log4J calls are used throughout the application, with some examples being seen in the main server class:

<https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2002/Hibernate/mpfinder/src/com/cit360/mpfinder/server/MissingPersonFinderServer.java>

The actual log can be found here: <https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2002/Hibernate/mpfinder/hibernate.log>

**Persistence to the Database via Hibernate**

The Server application receives a JSON object from a client application, parses it, and then, using the hibernate framework, it persists a Missing Person (Person) object to the database (**createMissingPerson method** -> <https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2002/Hibernate/mpfinder/src/com/cit360/mpfinder/server/controller/CreateMissingPersonHandler.java)>, or, it also retrieves a list of all the Missing Persons in the MySQL database that supports the application (**getMissingPersons** method -> <https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2002/Hibernate/mpfinder/src/com/cit360/mpfinder/server/controller/GetMissingPersonsHandler.java)>

**Threading**

The Sockets IO server example also uses threading to ensure that the server will continue to listen to multiple clients and they connect to the server:

<https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2002/Hibernate/mpfinder/src/com/cit360/mpfinder/server/MissingPersonFinderServer.java>

# Threads

As explained in the previous topic, the Sockets IO example uses threads to ensure that the server will respond to multiple clients connecting to the server. The following code implements the threading pattern:

**while**(!stopServer){

**try**(ServerSocket serverSocket = **new** ServerSocket(portNumber);){

**try**{

System.***out***.println("The Missing Persons Finder Server is now accepting requests on port " + serverSocket.getLocalPort());

Socket clientS = serverSocket.accept();

System.***out***.println("Connected client: " + clientS.getInetAddress());

Controller coms = **new** Controller(clientS);

**new** Thread(coms).start();

}

**catch**(IOException e){

**throw** **new** IOException("Unexpected error while processing client requests", e);

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

System.***out***.println("Exception: " + e.getMessage());

MissingPersonFinderServer.*getLogger*().error("Exception ->" + e.getStackTrace());

stopServer = **true**;

}

}

This code initiates a new thread of the sockets server each time a new client connects to it to initiate a request.

# Hibernate Libraries

In a previous submission, the feedback received was that additional technology integration should be used. This Sockets example, as explained in the previous topics, is fully integrated with the majority of the topics addressed in this class such as Sockets IO, JSON Serialization, Collections, Application Controller pattern, etc., and additionally, it uses a number of other topics not addressed, such as logging, properties files, singleton objects, etc.

The Hibernate pattern is utilized throughout the sockets server application to persist and retrieve data from the MYSQL database that supports the Missing Persons Application:

<https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2002/Hibernate/mpfinder/src/com/cit360/mpfinder/server/dao/PersonDAO.java>

<https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2002/Hibernate/mpfinder/src/com/cit360/mpfinder/server/model/Person.java>

<https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2002/Hibernate/mpfinder/src/com/cit360/mpfinder/server/model/Note.java>

<https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2002/Hibernate/mpfinder/resources/hibernate.cfg.xml>

<https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2002/Hibernate/mpfinder/resources/note.hbm.xml>

<https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2002/Hibernate/mpfinder/resources/person.hbm.xml>

# JUnit Test Creation

To test the Hibernate code, I created two tests that are used to confirm:

1. That the getMissingPersons method returns all the records from the database that it is supposed to. This is done by connecting to the database, and return a count of all the persons in the person table, and then compare it to the count of records returned by the getMissingPersons method.
2. That a record is successfully created in the database. This is done by ensuring the record to be created has a unique name, and after the record is created, execute a query that attempts to fetch that record, and comparing the resulting record, with the expected value.

<https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2002/Hibernate/mpfinder/src/com/cit360/mpfinder/server/junit/MissingPersonsFinderTests.java>

This test battery makes use of the @BeforeClass, and @AfterClass notations to create a database connection once, and then close it at the end of the tests.

Below is a screenshot of the test execution results in Eclipse:

