Understanding Repository - 02

Eurico Costa

CIT-360 W16

# Java Collections

I’ve been learning about java collections in the context of my personal android app and my understanding portfolio. The code I submitted this week for my portfolio that includes HttpURLConnection and Hibernate, have several implementations of collections, including Map, Set, List, ArrayList.

Implementations of these classes of the collections framework can be found here:

* HttpURLConnection, line 25: <https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2002/HttpURLConnection/src/HttpUrlConnectionDemo.java>
* MissingPersonFinder.java, lines 66, 203, <https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2002/Hibernate/mpfinder/src/com/cit360/mpfinder/MissingPersonFinder.java>

# Hibernate

I’ve started to research Hibernate and how to implement it in a Java program. This week I have conducted the following research:

<https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2002/Hibernate/Hibernate%20Research%20Paper.docx>

The following code is my first implementation of the hibernate framework and is tailored for my personal app project.

I have in the past worked on similar frameworks such as Hibernate. A few years ago, when I was working heavily with Microsoft .NET, and there were no similar frameworks around, I created my own framework, initially tailored for a project I was leading for a large Insurance Company back in Portugal. At the time, we wanted to create a web services layer in front of Siebel CRM 7.5, and basically creating a simple ESB to handle any type of transactions coming into Siebel, regardless of the format and transport used (SOAP web services, RESTful APIs, queues, etc.). the configuration for the transactions, as well as xslt transformations and maps between the incoming and outgoing transactions, and the canonical model we created were all in a MS SQL database, and instead of making this ESB tightly coupled to the database, I created a framework that abstracted the ESB from the database, so that we could easily make the ESB more portable and depending on the customer’s database preference, use either Oracle, or MS SQL, or anything else. We ended up creating support for both MSSQL and Oracle DB at the time.

So, this to explain that I understand the importance of having frameworks such as Hibernate to abstract these types of details from the application’s core. Developers spend way too much time writing code to do the same things such as database access, over and over. These frameworks remove the need to do so.

The code I implemented connects to a MySQL database on my ISP that I will use as part of my personal app project. It also implements error handling, logging via log4j to file, logger, and console, and uses Maven as well.

It implements a pojo for the Person and Note entities, and a PersonDAO class for the CRUD methods. All methods are demonstrated by executing the class MissingPersonFinder.java, and it can be found here:

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

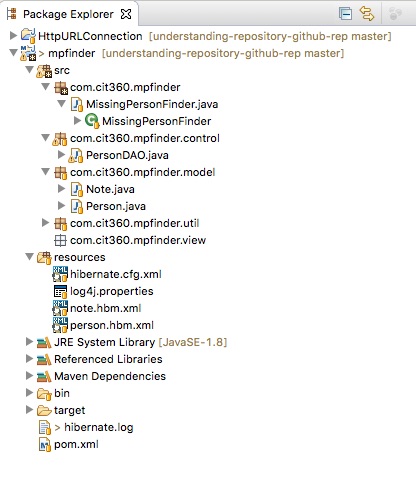
Main application folder can be found here: <https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/tree/master/Understanding%20Portfolio%20Submission%2002/Hibernate/mpfinder>

The code also implements the following patters:

* Singleton pattern for the SessionFactory (com.cti360.mpfinder.util.HibernateUtil class)
* The proxy pattern
* The Factory pattern

I created this code using the classical mapping file method. The next iteration, that will be as part of my app, I will use the annotations method.

Below is the structure of the application:



Although it was not my turn to present on this topic, in the last team meeting I presented my code since everyone was interested in learning more about Hibernate and no one besides me had yet delved on it. My presentation on my code can be found here:

<https://youtu.be/rUpbNWefXeM?t=2573>

# HttpURLConnection

For this topic I researched the HttpURLConnection class online and through a couple of Java books and prepared a presentation for my team meeting. For this presentation I wrote code that examines the contents of an Http request made via the HttpURLConnection class, examines its hearders, and diplays the content of the data retrieved. The code makes a call to an API in Weather Undergroung, and through a GET method, passes across the parameters required to get current weather conditions in my hometown of Alpine, UT.

As mentioned in the collections topic, it uses collections to examine all the header properties, and in addition, it uses IO streams to examine and display the result of the call, which in this case, is a JSON object. It also manipulates the JSON object to properly display the results, and uses proper error handling as well.

The presentation on this topic can be found here:

<https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2002/HttpURLConnection/HttpURLConnection.pptx>

The Java class that implements this code can be found here:

<https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2002/HttpURLConnection/src/HttpUrlConnectionDemo.java>

My presentation to the team can be found here:

<https://youtu.be/rUpbNWefXeM?t=1274>