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, which is going to be the app I presented in my pitch (<https://youtu.be/Al3K-SHBg_0)>, which is an app that will be used to report and find missing persons as a result of natural disasters. When the application is loaded, it will load to the screen a collection of records of missing persons, either that I am tracking, or that have recently had some sort of activity in the application, such as a new entry, or a record that received some sort of update.

## Research

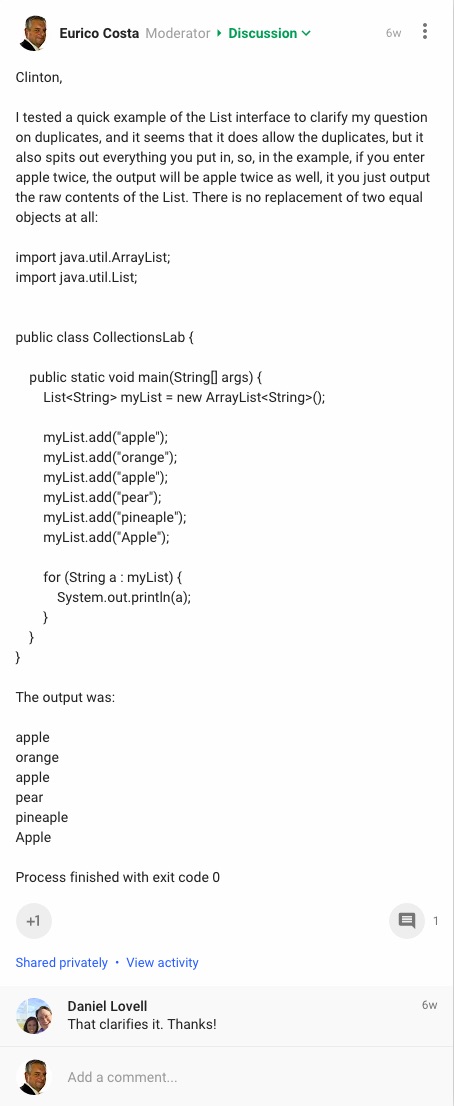
A collection is an object that groups multiple elements into a single unit. Collections are used to store, retrieve, manipulate, and communicate aggregate data. They represent data items that form a natural group, such as a collection of cards in a flash card study app, an email folder (a collection of emails), or a list of missing person records (as in the example of the app I am working on).

Java implements a Java Collections Framework that standardizes the way in which groups of objects are handled by programs. Collections were added in Java release J2SE, 1.2. This implementation was an improvement on the previous versions of what were the previous implementations of what could be called collections: Dictionary, Vector, Stack, Properties.

The modern collections framework is a highly optimized and high-performance implementation that shares a common framework that allows dynamic arrays, collections of different types of objects, and collection types that are also extensible. The list below is just a fraction of all the types of collections supported by the framework, and only reflects the most commonly used:

* ArrayList
* Dictionary
* LinkedList
* Properties
* TreeMap
* Vector
* Deque
* Enumeration
* List
* Map
* NavigableMap
* Queue
* Set
* SortedSet/SortedMap

In the first meeting we had as a team this semester, my colleague, Clinton, presented on the subject of Java collections. I was intrigued with something he mentioned in his presentation regarding regarding the List interface and how it allowed duplicates of objects in the list, and created the code below, with accompanying post to clarify the issue and confirm that, indeed, it allowed duplicates:



# 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.

# Sockets: Client/Server

This is one of my favorite domains in technology: anything that has to do with integration, so, I picked this topic from the list of topics that were available to the team.

To explain and teach the topic of Java Sockets and client server implementations, I used the following presentation:

<https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2001/sockets/SocketsIO.pptx>

To explain to my team the usage of sockets, I created an example program using a client/server implementation of sockets. The example is a client application that connects to a server to request weather details for a given location.

In addition to the sockets code sample, I decided to include JSON in my example, since it would be a common type of technology integration, using JSON to transmit data between a client and server application using sockets to communicate. Because native JSON support in Java will only be available in Java 9, I had to use the 3rd party Jason.org java JSON library that is referenced in the code.

The code also includes proper error handling. The files that implement this example can be found here:

Client: <https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2001/sockets/SocketsWeatherClient/src/SocketsWeatherClient.java>

Server: <https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2001/sockets/SocketsWeatherServer/src/SocketsWeatherServer.java>

To support the Server API call, the following Classes are also used:

<https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2001/sockets/SocketsWeatherServer/src/SocketWeatherProtocol.java>

<https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2001/sockets/SocketsWeatherServer/src/Weather.java>

The following link is a link to the sharing presentation where I presented on the topic plus the example implementation: <https://youtu.be/MeXdRb5NoJU>

The sockets topic has been discussed with my team on several occasions. The link below is for an excerpt of a discussion on the topic of sockets when my team and I were discussing the proposal for the team app: <https://youtu.be/ljViA9TU0SU?t=1396>

# JUnit

My most recent topic was on JUnit. On this topic, I created both a sharing presentation, which can be found here: <https://youtu.be/R-2bSFNYXFg>, and also presented about the topic in a team meeting, that can be found here: <https://youtu.be/c-SAsTOlYXg?t=1235>, with a discussion after on how to use JUnit.

The presentation can be found here: <https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2001/JUnit/JUnit.pptx>

The actual code used can be found in the links below:

Example Class: <https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2001/JUnit/JUnit_Example/src/junit_example/JUnitExample.java>

Test: <https://github.com/Costa-Eurico/CIT-360-W16-Understanding-Portfolio/blob/master/Understanding%20Portfolio%20Submission%2001/JUnit/JUnit_Example/test/junit_example/JUnitExampleTest.java>

The test class basically uses one example of all the functionalities available in the JUnit unit test framework, namely:

* Annotations (all annotations are used to exemplify the power of the framework)
* Assertions (all available assertions are used, with simple examples of its usage)