Refactoring Assignment 501 October 13, 2016

## By Austin Wattling

The code I’m using is from my networks class CPSC 441. I had 4 projects due on October 14th so I decided to make this one project a 2 in 1 deal and create a nice piece of refactored code fulfilling both sets of requirements. The program itself connects to a URL address and downloads what is specified only if that has not been modified since it was last saved. When I first wrote the program it was in my opinion pretty well done so in order to get 5 strong refactors out of it I had to “stink” it up a bit. I used Git as my version control system mainly using it through the eclipse IDE. The initial import and final product are included with the submission.

Refactor 1: Removing dispensable lazy class. The class FileIO was quite small and low in the hierarchy. The class above it Catalog was also quite small. To solve the stink of “lazy” classes I sucked up the FileIO class into the catalog class and collapsed the hierarchy. To apply the refactor I found the locations where FileIO was used by Catalog and put the appropriate code in place to keep the functionality the same. In doing so I did not create repetitive code pieces, removed lines of code and made the program easier to understand. The code was tested by calling the same methods in the parent class and ensuring the functionality remained the same. Included is a sample of what I did. The lines in red were removed and the lines in green were pulled from the FileIO class.

public class Catalog

{

HashMap<String, Date> catalog;

FileIO IO = new FileIO();

HashMap<String, Date> catalog = new HashMap<String, Date>();

@SuppressWarnings("unchecked")

public Catalog()

{

catalog = IO.getCatalog();

if(catalog == null)catalog = new HashMap<String, Date>();

try

{

File toRead=new File("Catalog");

FileInputStream fis=new FileInputStream(toRead);

ObjectInputStream ois=new ObjectInputStream(fis);

catalog =(HashMap<String,Date>)ois.readObject();

ois.close();

fis.close();

}

}

Refactor 2: Pulled out Class TCPClient from the bloated class UrlCache. The class UrlCache was fairly large and needed to be slimmed down as it took on too many things. I did this by creating the Class TCPClient to handle all HTTP protocol and internet retrieval. I found the methods in UrlCache that made the most sense in TCPClient and moved them there. By extracting the class TCPClient I was able to make the design of the program much easier to understand and link meaningful methods together in the new class. The code was tested again by simply running the program in a variety of states and insuring the functionality remained the same. This refactor enables me to refactor further as the methods handling similar things are collected in one place and are easier to mess with. The code included demonstrates the changes to getObject(url) as a result of the refactor.

public void getObject(String url) throws UrlCacheException

{

if (catalog.getTime(url) == null)

{

noTimeDownloadURL(url);

}

else

TCPClient client = new TCPClient();

if (catalog.getTime(url)==null)

{

timeDownloadURL(catalog.getTime(url), url);

client.noTimeDownloadURL(url);

}

catalog.AddToCatalog(url, getLastModified());

else client.timeDownloadURL(catalog.getTime(url), url);

catalog.AddToCatalog(url, client.getLastModified());

}

Refactor 3: Removing the stink of duplicate code and accidently removing a case of shotgun surgery. The newly created class TCPClient had two very similar methods. One that connected to the internet and retrieved the header and data as necessary only if a time was given called “timeDownloadURL(Date,URL)” and a method “noTimeDownloadURL(URL)” that would do the same with the exception of no time was given and the internet content was downloaded regardless. To solve this problem I created one Method called conditionalDownloadURL( URL, Date) that contained an if statement to check if the time passed in was null; If it was open the connection as it was there was no time entered. This refactor removed a ton of duplicate code and accidently removed a potential case of shotgun surgery. Now that all the connection opening was in one place if I needed to change it was all in one place rather than strewn across two different methods. This also made the next refactor almost more nessary than it was before. Testing was making sure the method behaved properly when given a time and not given a time. Included was the changes made to the connection part of the program including the newly created if statement that allowed to consolidation of the two methods.

String [] date = getTime.toString().split(" ");

out.println("GET " + path + " HTTP/1.0");

out.println("If-Unmodified-Since: " + date[0] + ", " + date[2] + " " + date[1] + " " + date[5] + " " + //

if (getTime != null)

{

String [] date = getTime.toString().split(" ");

out.println("GET " + path + " HTTP/1.0");

out.println("If-Unmodified-Since: " + date[0] + ", " + date[2] + " " + date[1] + " " + date[5] +//

}

else

{

out.println("GET " + path + " HTTP/1.0");

}

Refactor 4: Reduction of bloated long method conditionalDownloadURL( URL, Date) in class TCPClient. This method was huge and took on a lot of functionality. It needed to be broken down into smaller chunks. I extracted two methods from the class. One method that simply parsed the URL into its various pieces and another that was in charge of opening the socket connection and returning the socket connection to the method that called it. The functionality of method conditionalDownloadURL stayed the same. The code was much more readable. It eliminated the search to find the part of the method you wanted to change and rather allowed changing of individual methods. To test I would run each method separately and make sure it ran and returned everything as expected. No code example will be included here but it’s easy to see the changes in the final copy of the file TCPCLient.

Refactor 5: Renames and commenting. Parts of the program were left to the imagination of some readers as nothing was commented and names were unclear or could have been better. For example, the name of the file saved through the catalog was called “catalog” when the more widely used name “cache” was more descriptive for the purpose it was serving. As I’m handing in this code for another project it was important I had everything is well commented and named. This makes the code more readable to another programmer and in my mind a total refactor in itself. “Stinky” code is code that is hard to understand and sometimes comments are necessary to make things much easier to understand thus mitigating the stink. Included in this refactor were a couple of small code line moves that should have removed a little unnecessary overhead. Things like initiating a list object that may never be used when it you could only initialize within the if statement where it’s used once it’s for sure needed. Testing was quite simply running the same tests as before and insuring no new bugs appeared as the result of a semicolon being moved or bracket being deleted etc. I include no code here as these kinds of things are pretty intuitive and easily seen in the final version of the code.

I have included a small number of tests on my TCPClient class. I ran out of hours to write proper tests for this as they would be quite in-depth deleting directories and files before running parts of my program. I just wanted to show I can use a testing framework .

commit b8073e9ecd42609b2bbd938b364d78d137a327f9

Author: Austin Wattling <Austin Wattling@Cindy>

Date: Thu Oct 13 23:32:10 2016 -0600

Various local name changes. Changed the catalog file save to more

apporiate name "cache" save. Changed all "URL" String tags to "url"

instead of having a mix. Final commenting and cleaning on project

commit ee98a2e80ad39e8005a6d623a1c12cbad0f88516

Author: Austin Wattling <Austin Wattling@Cindy>

Date: Thu Oct 13 21:04:14 2016 -0600

Divided up the workload of the conditionalDownloadURL method in class

TCP Client. This method was extremely large and was divided into a

method specifically for opening a socket connection a method to parse

the URL and the conditonalDownload Method now only handles the actual

HTTP IO.

commit aa1a569e3eb97c35e94124877e1792ee54c863b1

Author: Austin Wattling <Austin Wattling@Cindy>

Date: Thu Oct 13 20:49:56 2016 -0600

Removed duplicate code in TCPClient. This was very repetitive code that

was solved using a simple if statement. Resulted in the loss of the

TimeDownloadURL methods for one method ConditionalDownloadURL. Also

unintentionally solved a little bit of shotgun Surgery.

commit 6fe2764928ceca1ee77ba2eee0b0da27be0cf874

Author: Austin Wattling <Austin Wattling@Cindy>

Date: Thu Oct 13 20:34:01 2016 -0600

Slimed down the bloated large class that was UrlCache and created the

class TCPClient the should handle the all the HTTP protocol and Internet

retrieval stuff.

commit 7fe0a76248b69b038bdbef9c7d1ff24eee391520

Author: Austin Wattling <Austin Wattling@Cindy>

Date: Thu Oct 13 20:07:37 2016 -0600

Removed Lazy class FileIO and allowed class catalog to handle all the IO

stuff that FileIO did before.

commit fa7dbd17322eb19b9c4f07c5fc8045f94176d3b0

Author: Austin Wattling <Austin Wattling@Cindy>

Date: Thu Oct 13 20:01:53 2016 -0600

Initial Import