**Lab 4**

Inner classes and Exceptions (due Feb. 26)

Purpose

This lab is to declare an inner class and use exceptions. This lab is a continuation of Lab 1, using the URL class. The UML class for the updated URL class is given, with new methods and variables. I am also giving you the public static void main method that tests your implementation of the new methods. However, the steps given show you how to use Eclipse’s suggested solutions to help speed up the development of new code.

Background

Look at the initial URL: "https://www.google.ca/search?q=java+string&sourceid=chrome&ie=UTF-8".

There are several pieces of information encoded in this string. First, the protocol is the string that comes before “://”. In the above string, the protocol is “https”, but many web pages use “http” instead.

After the protocol is the server address. It comes between the “://” string and the next “/” character. In the example, the server address is [www.google.ca](http://www.google.ca).

The server path is the directory on the server where the document is located. It might not be part of the string so you have to handle that case. If it is there, then it comes after the server address, and the “/” character. In the example, the path is “search”.

The search parameters, if they exist, come after the “?” character, and are separated by “&” characters. For example, “?param1=value1&param2=value2&param3=value3”, has parameters: “param1=value1”, “param2=value2”, “param3=value3”.

Parameters and values are separated by the “=” character. For our original example: the parameters ***q*** has the value “*java+string*”, the parameter ***sourceid*** has the value “chrome”, and the parameter ***ie*** has the value “UTF8”.

We will learn how to use some useful methods in the String class: indexOf(String), substring(int start, int end) to help separate the parts of a query string and set the appropriate portions of the HTTP\_Request object.

Steps

1. Pull the latest source code from Github in the CST8284\_W16 project. There should be a Lab\_4 package. Create a new Eclipse project for Lab4, and copy and paste the Lab\_4 package into your new project. **DO NOT** do your lab work in the CST8284\_W16 project because pulling code from Github afterwards will **overwrite your work**.
2. Look in the Lab\_4 package, there are sample HTTP\_Request and Test classes. If you look at the main method in the Test.java class, there are a lot of red underlines indicating error in the code. For example, line 10 has new HTTP\_Request(searchQuery, 4); underlined. It is because the constructor for HTTP\_Request with a string and int parameter doesn’t exist. If you put your mouse over the red line, a pop-up window should appear saying “The constructor HTTP\_Request(String, int) is undefined. There is another section of the pop-up window that suggests 5 possible quick fixes: 2 are to remove parameters from the constructor, 2 are to change existing constructors to have the right number of parameters. The last suggestion is to create a new constructor with String, int parameters. The String should be the URL, and the int should be the maximum number of parameters that you can store in your HTTP\_Request object. An int of -1 should mean that there is no limit. Click your mouse on the last suggestion to create a new constructor. Both constructors should throw MalformedQueryExceptions if the protocol or address are not found.
3. Put your mouse over the MalformedQueryException error and look at the pop-up.

The correct error is that the class doesn’t exist so select the quick fix that creates the class. Notice that because the MalformedQueryException is in a catch() block, Eclipse can guess that it should be a subclass of Exception. Click Finish to create the class.

1. The next errors: request.getProtocol(), request.getServerAddress(), request.getServerPath(), request.getSearchParameters(), request.includesParam() are caused because these methods don’t exist yet in your HTTP\_Request class. Select the quick fix to create the method in HTTP\_Request. Do the same for getValueForParam(), setValueForParameter(), and addParameter().
2. Click the quick fix to create the ParameterNotFoundException class and add the getParameter() method, and then create the ParameterArrayFullException class.
3. In your HTTP\_Request class, create a private inner class called NameValuePair that has both a name and value String variables.
4. Go to the new constructor that you created and write an algorithm that will scan through the query string to find the individual parts. First, find the protocol type. Use the indexOf(“://”) method to calculate where in the request string the “://” string begins. It should be 5 because “https” comes first, followed by “://”. The indexOf() method returns -1 if the string wasn’t found. In that case, the constructor should throw an exception because the string “://” must appear in the URL. Look at the main method to tell which exception should be thrown, and give an appropriate message saying why the exception was thrown, in this case ‘The protocol could not be determined’.
5. Once you have found the protocol, the next portion should be the server address. It comes between the “://” and the next “/” characters. There is a version of indexOf() that looks for a string starting at a given position: indexOf(String str, int fromIndex). Since you should have found where the “://” starts, you should look for the indexOf(“/”) starting from the previous position.

://www.google.ca/search?q=java+string&sourceid=chrome&ie=UTF-8.

1. If there is no “/”, then indexOf(“/”) will be -1, and all of the other values: server path and query parameters should be set to null. Otherwise, the server path is the substring from the first “/” until the end of the string, or the first “?” if there is one.

/search?q=java+string&sourceid=chrome&ie=UTF-8

The ? represents a query, and everything that comes after are parameters separated by &, where ParameterName=Value.

?q=java+string&sourceid=chrome&ie=UTF-8

1. Write an algorithm that separates each of these parts of the search query, and sets the server name, server path, and query string name/value variables of the HTTP\_Request object.
2. The name/value variables should be stored in order in an ArrayList<NameValuePair>. The addParameter(String name, String value) method should add a new NameValuePair object to the ArrayList. If the ArrayList size is equal to the maxSize, then it should throw a ParameterArrayFullException, with an appropriate message.
3. The setValueForParam(String *name*, String *value*) should iterate through the ArrayList and look for a name that matches, and overwrite the current string with the new string *value*. If the parameter name doesn’t exist, then it should throw a ParameterNotFoundException with the message explaining which parameter name was being set.

Once this runs properly without crashing, set a breakpoint on line 7 in Test.java and demonstrate the flow of the program to the lab professor. This lab is worth 16 marks:

Line 12 HTTP\_Request constructor does not throw MalformedQueryException +1

MalformedQueryException class exists +1

lines 19-22 return correct values +4

line 24 includes parameter works properly +1

lines 26-28 pass without throwing exception +3

lines 30&34 classes exist for ParameterNotFoundExeption and ParameterArrayFullException +2

line 46 exception is thrown properly +1

Line 52 does not throw MalformedQueryException +1

line 65 throws an exception +1

line 73 throws an exception +1