Jackson State University
Department of Computer Science
CSC 435/524 Computer Networks, Spring 2014
Instructor: Dr. Natarajan Meghanathan
Project # 1: Implementation of a Remote String Processor Application

Due: February 21, 2014 (Hard deadline: No postponement) **Max. Points:** 100

In this project, you will implement a "Remote String Processor" application comprising of two programs (processes) using connectionless sockets (terminology: sender/receiver processes) as well as using connection-oriented sockets (terminology: client/server processes). The two implementations (using connectionless and connection-oriented sockets) are independent from each other and are worth for a maximum of 50 points, each.

String Methods: Use the following integer codes for the different string methods. For more information on each of these String methods and their return type, refer to the Java API.

Integer Code	*String Method*	*What it does?*
1	string1.compareTo(string2)	Returns a positive integer if string1 lexicographically follows string2 (e.g., string1 = "Bob", string2 = "Apple") Returns a negative integer if string2 lexicographically follows string1 (e.g., string1 = "Apple", string2 = "Bob") Returns 0 if the two strings are the same
2	string1.concat(string2)	Appends string2 to string1 and returns the concatenated string
3	string1.endsWith(string2)	Returns true if string1 ends with string2; else return false
4	string1.startsWith(string2)	Returns true if string1 starts with string2; else false
5	string1.indexOf(string2)	Returns the integer index of the first occurrence of string2 in string1; returns -1 if string2 does not occur in string1

Synopsis: When the client/sender process is started, it should display a menu of the valid string methods along with their integer code that can be executed on a string object. The user inputs the two strings (string1 and string2) and the integer code value of the method to be executed. If the user inputs integer code outside the range of 1...5, the client/sender process should print an error message and terminate.

If the user enters a valid input for the integer code (i.e., from 1...5) as well as the two strings, the client/sender process sends the two strings and the integer code to the server/receiver process. Depending on the integer code, the server/receiver process executes the matching string method (according to the above table) and sends the appropriate return value (a string or an integer or a Boolean value) depending on the method executed.

The client/sender should process the return value depending on the integer code value input by the user, and appropriately print the output (see the section below for sample output statements).

Sample Output Statements

<X> indicates the value of variable X to be printed

Integer Code	*String Method*	*Sample Output Statements (depending on the value of the strings)*
1	compareTo	<string1> lexicographically follows <string2> <string2> lexicographically follows <string1>

		<string1> equals <string2>
2	concat	<string1> concatenated with <string2> is <concatenated string>
3	endsWith	<string1> ends with <string2>
		<string1> does not end with <string2>
4	startsWith	<string1> starts with <string2>
		<string1> does not start with <string2>
5	indexOf	<string2> first occurs at index <index value> of <string1>
		<string2> doest not occur in <string1>

You are free to choose your own design/logic to implement your programs. You need to clearly explain (in the report and the video) the approach you took for the implementation.

Port Number: Let the server/receiver process to run on a port number corresponding to the last 4 digits of your J#. If the last 4 digits of your J# is less than 1024, add 10000 to it, and run on the resulting port number.

Server/Receiver Design: Make sure, your ***server/receiver is running continuously and can handle requests from infinitely many clients/senders*** (one at a time) and your client/sender program should be run each time you want to do string processing. The server/receiver process can be stopped by pressing CTRL+Z or CTRL+C (by doing so, the socket will also be closed at the server/receiver side. So, do not worry about closing the socket if you are running the server/receiver in an infinite loop).

For each client/sender served, the server/receiver should locally print (on screen) the IP address and port number (of the client/sender) from which the request for string processing came.

Things/Issues that you may encounter and need to use

trim( ) method: If you are sending a string (or a concatenated list of strings) across a socket, make sure to run the trim( ) method on the received string before processing it further for any operations. For example, if receivedString is the String object received, then you can call

String trimmedString = receivedString.trim( );

StringTokenizer class: You may need to use the StringTokenizer class and its methods to put together the strings and the integer opcode and send as one single string from one side and extract them at the other side. Refer to the example (Java code and video explanation: http://www.youtube.com/watch?v=jMbbvMeZuDc) posted along with this project description to know more about the use of StringTokenizer.

Datagram Socket Implementation: In the sender-receiver implementation using datagram sockets, you may encounter a scenario where you declare a byte array (*recvBuffer* in code below) of a default larger value to hold any return value from the other end. In that case, after you read the byte array as a string (*rString*), transform it to another byte array (*rString*.getBytes( )) and construct a new string (*recvdString*) out of that byte array, and then trim it - to weed out the additional spaces inserted in the original received string (corresponding to the default byte array) and the white spaces resulting from the second byte array. You could consider to use a code similar to this below:

```
// to receive a message
int MESSAGE_LEN = 100;  // default value
byte[] recvBuffer = new byte[MESSAGE_LEN];
DatagramPacket datagram = new DatagramPacket(recvBuffer, MESSAGE_LEN);
mySocket.receive(datagram);
String rString = new String(recvBuffer);  // first string as received
String recvdString = new String(rString.getBytes()).trim();
// extract the return value from recvdString  (it will have the exact value returned from the other side)
```

Where to run the programs: You could run these programs on your personal computer, using the localhost option (put your client/sender and server/receiver programs in two separate folders).

If you do not have access to a personal computer on which you can run the Java programs and/or do a desktop recording, you can do the project and/or do the desktop recording of your explanation of the project on a computer at the Computer Networks Systems and Security lab on campus (Room 110 at J. Y. Woodard Building).

What to Submit:
(1) **A video file** (either one of these formats: .mp4, .wmv, .avi) that is generated by desktop recording your explanation of the working of your program and the logic/approach you took to implement the "Remote String Processor" to satisfy the design requirements (including the error different types of responses returned). You should display the program(s) on the desktop and walkover the different sections of your client/sender and server/receiver code as well as explain the execution flow of the programs. You should also record demonstrating the working of your programs: run the server/receiver process that runs infinitely and run at least five client/sender programs one after the other, each sending different string values and entering a different integer code operation. **Note that even though I am not specifying a minimum time for your video, your video explanation is expected to last at least for 8-10 minutes and should cover all of the above required explanations.**

Note that the contents of the desktop/programs captured through your video should be clearly readable.

Submit the video through Google Drive (using your JSU email address) and send the link via email to natarajan.meghanathan@jsums.edu

You could try using one of the **desktop recording software** (or anything of your choice):
CamStudio: http://sourceforge.net/projects/camstudio/files/legacy/
Debut: http://www.nchsoftware.com/capture/index.html

(2) A **report** featuring the following: (i) Your code for the client/server (for connection-oriented sockets) and for the sender/receiver (for connectionless sockets) and snapshots of at least five client/sender processes (placed suitably on your monitor screen) showing the passage of inputs and the result obtained as well as a snapshot of the server printing the IP address/port number of the different clients/senders from which the inputs for string processing were received. (ii) A paragraph (at least 200 words) explaining the design/logic behind your implementation.

Submission:
(1) Video file shared (to: natarajan.meghanathan@jsums.edu) via Google Drive
(2) Hardcopy of the report, submit in class
(3) Hardcopy of the report, emailed to the instructor: natarajan.meghanathan@jsums.edu. Clearly mention the Subject of the e-mail as "CSC 435, Spring 2014, Project 1 Submission" or "CSC 524, Spring 2014, Project 1 Submission" depending on your classification. The e-mail should be sent from an email address that clearly specifies your name.