CS242: Advanced Programming Concepts in Java
Programming Project: Clack

Overview

In this multi-part team programming project, you will design and implement Clack (short for Clarkson Slack), an instant messaging application akin to Slack that runs as a standalone application. Since this is a team project, you must have a teammate prior to starting the project.

Introductions and Practice for Git setup

Introductions. Plan a meeting within your team. This interaction is not graded, but is highly recommended to get into the team spirit for a successful virtual collaboration for the project.

Practice for Git setup. Follow the slides in Lecture 2: Software Project Development, specifically, Xinchao's presentation on Software Project Development: Tools, to practice 2-person team setup in GitHub. Create a HelloWorld project to perform this practice. This practice is graded (rubric can be found at the end of this write-up).

Part 1: Class Hierarchy

Since you are all working toward being good software developers, the first step in implementing a complex software project is determining the structure and hierarchy of all classes involved in the project. While you will not add all classes right away in this project, you will get started with the most important ones.

Clack follows a **client-server model** (which we will discuss in more detail later in the course). In a client server model, data is transferred across a network between two computers, a server computer and a client computer. In your application at the end, you will have multiple client computers, and a single server computer. **Start soon!** As you will notice, there is quite a bit of set up to do in Part 1.

Your classes will consist of the following:

- ClackClient: the class for the client,
- ClackServer: the class for the server,
- ClackData, FileClackData, and MessageClackData: classes for the data transferred between the client and server.

Tasks

In Part 1, you will achieve the following tasks:

1. **Create the Git setup for your project**
2. **Develop the class interface for all classes (i.e., class and method signatures), with correct package hierarchy (refer to the section titled "Package Hierarchy")**

3. **Implement majority for the methods for the data classes ClackData, FileClackData, and MessageClackData,**
4. **Implement some of the methods for the classes ClackClient and ClackServer,**
5. **Write the test classes TestClackData, TestClackClient, and TestClackServer to test all implemented methods,**
6. **Have Javadoc comments for all classes and methods.**

Note: there is a bit of an abuse of notation here. The word data is being used to refer to information transferred between the clients and the server. But we know that "data" refers to the instance variables of an object. To avoid confusion, we will refer to the data of a class as instance variables throughout this project, and the data transferred between the client and the server as just data.

Before you start, follow the steps you practiced in the Git setup practice to now create the repository for your actual project. Decide amongst yourselves who will be Person A (i.e., the creator of the repository) and Person B.

Class Structure

Class ClackData is a superclass that represents the data sent between the client and the server. An object of type ClackData consists of the user name of the client user, the date and time at which the data was sent and the data itself, which can either be a message (MessageClackData) or the name and contents of a file (FileClackData). MessageClackData and FileClackData are both children of ClackData. Note that ClackData should not be instantiable, but MessageClackData and FileClackData should be. **All data classes will be in package "data".**

Remember to make use of the three advantages of object-oriented programming: encapsulation, inheritance, and polymorphism.

Class structure for ClackData	
username	String representing name of client user
type	An integer, refer to section labeled "About the type variable"
date	Date object representing date when ClackData object was created
ClackData(userName, type)	Constructor to set up userName and type, date should be created automatically here
ClackData(type)	Constructor to create anonymous user, whose name should be "Anon", should call another constructor
ClackData()	Default constructor, should call another constructor. What should "type" get defaulted to?
getType()	Return the type

getUserName()	Return the user name
getDate()	Return the date
getData()	**Abstract** method to return the data contained in this class (contents of instant message or contents of a file)

About the type variable:
The variable type represents the kind of data exchanged between the client and the server. It is meant to enable the server to provide the client with targeted service. The variable type can take on one of the following four constant values:

CONSTANT_LISTUSERS: give a listing of all users connected to this session
CONSTANT_LOGOUT: log out, i.e., close this client's connection
CONSTANT_SENDMESSAGE: send a message
CONSTANT_SENDFILE: send a file

ClackData objects of type CONSTANT_LISTUSERS, CONSTANT_LOGOUT, and CONSTANT_SENDMESSAGE, will get instantiated as MessageClackData. ClackData objects of type CONSTANT_SENDFILE will get instantiated as FileClackData.
CONSTANT_LISTUSERS should be initialized as a *public* constant in ClackData using the 'final' keyword, and its value should be set to 0. Similarly CONSTANT_LOGOUT, CONSTANT_SENDMESSAGE, and CONSTANT_SENDFILE should each be public constants, and their values should be 1, 2, and 3. In future parts of the project, you will be implementing functionality for sending files and messages, logging out, and listing users by checking against these constant values. Here, you just need to declare them in preparation for the future, but you do not need to use them.

Class Structure for MessageClackData: inherits from ClackData	
message	String representing instant message
MessageClackData(userName, message, type)	constructor to set up username, message, and type, **should call super constructor**
MessageClackData()	default constructor, should call another constructor
getData()	implemented here to return instant message
hashCode()	should be correctly overridden
equals():	should be correctly overridden
toString()	should be overridden to return a full description of the class with all instance variables, including those in super class

Class structure for FileClackData: inherits from ClackData	
fileName	String representing name of file
fileContents	String representing contents of file
FileClackData(userName, fileName, type)	Constructor to set up username, fileName, and type, **should call super constructor,** fileContents should be initialized to be null
FileClackData()	Default constructor, should call super constructor
setFileName(fileName)	Sets the file name in this object
getFileName()	Returns the file name
getData()	Implemented here to return file contents (currently null)
readFileContents()	Does not return anything, **for now it should have no code, just a declaration**
writeFileContents()	Does not return anything, **for now it should have no code, just a declaration**
hashCode()	Should be correctly overridden
equals()	Should be correctly overridden
toString()	Should be overridden to return a full description of the class with all instance variables, including those in super class

The ClackClient class represents the client user. A ClackClient object contains the **user name**, **host name of the server connected to**, **port number connected to**, and a boolean designating **whether the connection is open or not**. Every computer has an address that you can connect to, and in the case of a server, this address represents the **host name** of the server, so called because it is the host, the main guy through whom all the little clients have their communication. Every computer also has ports to which you connect, these ports enable different applications to simultaneously connect to the network. You connect to a host server at a designated **port number**. The ClackClient object will also have two ClackData objects representing data sent to the server and data received from the server. **ClackClient should be in package "main".**

Class Structure for ClackClient Class	
userName	String representing name of the client
hostName	String representing name of the computer representing the server
port	integer representing port number on server connected to
closeConnection	boolean representing whether connection is closed or not
dataToSendToServer	ClackData object representing data sent to server
dataToReceiveFromServer	ClackData object representing data received from the server
ClackClient(userName, hostName, port)	Constructor for username, host name, and port, connection should be set to be open. Should set dataToSendToServer and dataToReceiveFromServer as null.
ClackClient(userName, hostName)	Constructor to set up port to default port number 7000, default port number should be set up as constant, this constructor should call another constructor.
ClackClient(userName)	Constructor that sets host name to be "localhost" (i.e., the server and client programs run on the same computer)
ClackClient()	Default constructor that sets anonymous user, should call another constructor.

start()	Does not return anything, **for now it should have no code, just a declaration**
readClientData()	Reads the data from the client, does not return anything, **for now it should have no code, just a declaration**
sendData()	Sends data to server, does not return anything, **for now it should have no code, just a declaration**
receiveData()	Receives data from the server, does not return anything **for now it should have no code, just a declaration**
printData()	Prints the received data to standard output, **for now it should have no code, just a declaration**
getUserName()	Returns the user name
getHostName()	Returns the host name
getPort()	Returns the port
hashCode()	Should be correctly overridden
equals()	Should be correctly overridden
toString()	Should be overridden to return a full description of the class with all instance variables

The ClackServer class is a blueprint for a ClackServer object that contains information about the port number that clients connect to, a boolean representing whether the server needs to be closed or not, and ClackData objects representing data sent to and received from the client. The server class does not need to know the host name (as the server program runs on its own computer), it just needs to know what port the clients connect to. In our application, all clients will connect to a single port. **ClackServer should be in package "main".**

Class Structure for ClackServer class	
port	integer representing port number on server connected to
closeConnection	boolean representing whether connection is closed or not
dataToReceiveFromClient	ClackData object representing data received from the client
dataToSendToClient	ClackData object representing data sent to client
ClackServer(port)	Constructor that sets port number, should set dataToReceiveFromClient and dataToSendToClient as null.
ClackServer()	Default constructor that sets port to **default port number 7000**, default port number should be set up as constant, this constructor should call another constructor.
start()	Does not return anything, **for now it should have no code, just a declaration**
receiveData()	Receives data from client, does not return anything **for now it should have no code, just a declaration**
sendData()	Sends data to client, does not return anything, **for now it should have no code, just a declaration**
getPort()	Returns the port
hashCode()	Should be correctly overridden
equals():	Should be correctly overridden
toString()	Should be overridden to return a full description of the class with all instance variables

As you will see the server class mirrors the client class in interface to an extent. However, the implementation of the methods (which you will do in Part 2 onwards) will be specific to the functionality of a client and a server.

Test Classes:

You must write three test classes, TestClackData, TestClackClient, and TestClackServer. TestClackData should instantiate MessageClackData and FileClackData objects, and test all implemented functions in the data classes (whether they are in the superclass or in the subclasses). TestClackClient and TestClackServer should create ClackClient and ClackServer objects and test all implemented functions. Test classes should be in package "test".

What happens if you provide a negative value for a port number in TestClackClient or TestClackServer, or a null value for a user for any class? How do you think you can fix these issues (you do not need to fix them, just deliberate on how you will fix them).

Package hierarchy:

1. Data classes should all be in the package "data"
2. Client and Server classes should be in the package "main"
3. Test classes should be in the package "test"

Submission:

There are two required components to the submission:

1. **All code for the project:** Your submission must consist of **all .java code**. Your code must contain Javadoc comments, and your submission must also contain **the Javadoc in a doc/ folder**, and a short report written in Word, OpenOffice or a similar document editor. Since you will be using Git, if you use GitHub, you are free to use the README feature to write the report (however, that is not compulsory). **The report should have team member names clearly spelt out at the top**, and you should answer the following questions in the report:

    - In your test classes, what happens if you provide a negative value for a port number, or a null value for a user? How do you think you can fix these issues?

    Your submission must be posted to Moodle at the link "Project Part 1 Submission". In the text submission box, you are required to type in the link to your online Git repository. You are recommended to keep your Git repository private, and to add the TA Xinchao Song (xisong@clarkson.edu) to your repository users list, so that your assignment is graded. The Git repository must contain your entire code, Javadoc doc/ folder, and report. You must include your .java files stored under an "src/" directory in the correct folder hierarchy that emanates on using packages. Please do not include supporting project directories that arise when you use an IDE. Designate a single person from your team as the 'submitter', and submit through their Moodle account.

2. **A report containing an assessment of your contribution and your teammate's**

contribution: This report is a private report that **is not to be shared with your teammate**. Use one paragraph to discuss your contribution and one paragraph to discuss your teammate's contributions. This report should be uploaded to Moodle at the link "Project Part 1 Contributions". Each teammate should submit a report to their own Moodle account.

Grading Rubric (out of 5):

Task	**Grade allotment**
Git practice	.25
Correctly written ClackData	.5
Correctly written FileClackData	.5
Correctly written MessageClackData	.5
Correctly written ClackServer	.5
Correctly written ClackClient	.5
Correctly written and running TestClackData	.5
Correctly written and running TestClackServer	.5
Correctly written and running TestClackClient	.5
Javadoc comments, and Javadoc folder	.25
Report with all questions answered	.25
Contributions report	.25