Exercise 1

Before starting this exercise session, make sure you have read the problem statement document for the course's assignment. Some information required for this exercise, such as the format for saving user and message data in text files, is described there and is missing in this document. In this exercise session, you will implement the first two versions of the chat system: ChatSys 0.1 and ChatSys 0.2.

Configuring IntelliJ

Before starting to write any code, let's configure IntelliJ. Start by creating a new Maven project using Java 8. (If Java 8 is not installed, you can click "Download JDK" and select Java 8 from AdoptOpenJDK OpenJ9.) Then click next and name your project ChatSys. In the "artifact coordinates", you can change the groupid with "programming3", the artifactid to "ChatSys" and the version to "0.1".

Before starting, create a package "programming3.chatsys". You should put all your classes inside that package or a sub-package.

Also initialize a git repository inside your IntelliJ project. You can do this either using the command line or using IntelliJ's GUI. Ideally, you should regularly commit your changes so that you save your progress and can go backward if needed.

ChatSys 0.1 - Sending messages

Chat messages

Let's first create the class ChatMessage inside the package "programming3.chatsys.data" that will be able to read and save chat messages. Create the class with its attributes, getter methods and constructors to initialize them. Also implement the toString, equals and hashCode methods. You can use IntelliJ to generate those for you.

Once done, add two empty methods:

- format(): String
- parse(String formatted): void

The format method will return a formatted String representing the object while the parse method will set the attributes to the values of the formatted String given as argument. For the time being, return null for the format method and leave the parse method empty.

Create a test class in order to write unit tests for ChatMessage: ChatMessageTest. Create a test method for the format and parse methods. These can be created automatically with IntelliJ.

Implement the test methods created. Run the tests. These should fail as you haven't implemented the methods yet.

Now implement the format and parse methods. Run your tests. If they don't pass, it means there is a bug in your methods and you should fix them. Continue until all the tests pass and you are confident the methods work properly.

Now create a method save. The method takes a file as an argument and will save the object into that file. Make sure that the file is opened in “append” mode. This prevents the file from being completely erased every time we write inside! You can do this by creating first a FileWriter like this: “new FileWriter(filename, true)”. Then you can pass the FileWriter to a PrintWriter.

Don’t forget to write unit tests for the save method. Inside the test, save your results in a file “messages_test.txt”. After your test are executed, don’t forget to remove the temporary file. You can do this by defining a method “clean” inside your test class and use the annotations @AfterAll or @AfterEach.

Create a first commit with git to save your progress.

Database

Create a class “Database”. For the time being, the class will have one attributes referencing the file for the message database. This class has one public method: readMessages and returns a List of ChatMessages. It will be obtained by reading the database files. For the time being, simply return null.

Create a new test class “DatabaseTest” and one test method for readMessages method. Implement the test method. Run the test class and the test should fail.

Now implement the method. Use a BufferedReader to read each line of text and parse them using the parse method of ChatMessage.

Add to the Database a method addMessage that takes a ChatMessage as argument and adds it to the database file. The chat message added should have an id greater than all the ids of all the chat messages already in the database. If it doesn’t, the method should throw an exception.

One issue with our message text file is that the parsing won’t work if there is a line feed (i.e. “\n”) in any of the text fields. Moreover, the username can only contain letters, numbers and the underscore (_) character. Add the appropriate code in the parse and format methods to throw an exception if the String to parse or any of the String class attributes are not formatted properly. Add a unit test to check that the parse and format methods fail (i.e. throw an exception) when given an invalid input.

Make a new commit to save your progress.

Command Line Interface (CLI)

Now create two classes inside the package “programming3.chatsys.cli” with a static method “main” that will be used as a command line interface to read and save messages.

SendMessages do the following operations:

- Ask for a username.
- Check if the database exists and is not empty.
- If it exists and isn’t empty, read all the messages and find the chat message with the biggest id and save that id as the “last id”.
- If the DB doesn’t exist or is empty, save 0 as the last id.
- Then, use a Scanner in Java to ask the user to enter text.
- For each line of text entered, increment the last id by 1. Create a ChatMessage using the incremented last id as id, the current time for timestamp and the username of the authenticated user. Save it in the message database.

ReadMessages:

- Ask for a number n.
- Read the message database and display the n most recent messages. If n is 0, shows all the messages.

Note: asking for the number of messages to read (for ReadMessages) or the user information (for SendMessages) can be done either with a Scanner or using the command line arguments.

Make a new commit with git. Also create a tag with git and name it “0.1”

ChatSys 0.2 – Registering and authenticating users

User

Now create the class User (in programming3.chatsys.data) in a similar way that you created ChatMessage. Add the getter methods, toString, equals, hashCode, format, parse and save. Create a test class UserTest and add unit tests for format, parse and save. By default, the lastReadId of a User is 0. In the current version of ChatSys, it won’t be changed.

Add to the Database a method readUsers and its associated test in DatabaseTest. The readUsers method returns a Map<String, User> where each key is the username, the second one a List<ChatMessage>.

Make a new commit to save your progress.

CLI

Now create a class, named RegisterUser (in programming3.chatsys.cli), with a static main method. It should do the following:

- Ask for a username, full name and password
- Read the username database
- If the username is in the database, exit with an error message
- If the username is not in the database, save the information in the database

Now create a class SendMessagesAuthentication. This class is similar to SendMessages but adds a few operations (the text in bold represents the operations that are new in comparison with SendMessages):

- Ask for a username **and password**.
- **Reads the user database and search for the username.**
- **If the username it is not in the database or the password doesn’t match, return an error message and exit**
- Check if the database exists and is not empty.
- If it exists and isn’t empty, read all the messages and find the chat message with the biggest id and save that id as the “last id”.
- If the DB doesn’t exist or is empty, save 0 as the last id.
- Then, use a Scanner in Java to ask the user to enter text.
- For each line of text entered, increment the last id by 1. Create a ChatMessage using the incremented last id as id, the current time for timestamp and the username of the authenticated user. Save it in the message database.

Note: asking for the number of messages to read (for ReadMessages) or the user information (for SendMessagesAuthentication and RegisterUser) can be done either with a Scanner or using the command line arguments.

Maven

Finally we are going to generate a JAR file with Maven.

Open the pom.xml. There you should see an element <version>0.1</version>. Update the version number from 0.1 to 0.2.

On the right side of the IntelliJ, there should be a tab “Maven”, click on it and then in “lifecycle”, double click on “package” to run a maven build. This will create a “target” directory inside your project directory containing everything that Maven has built, including a JAR file that should be named “ChatSys-0.2.jar”.

Make a git commit and create a tag “0.2”.

ChatSys 0.3 – Unread messages

Finally, let’s add the option for a user to obtain only the list of message they haven’t yet read.

For that, we will add a method “getUnreadMessages(String userName): List<ChatMessage>” in Database. Also create a private method “updateLastReadId(String userName, int id)”.

getUnreadMessages should do the followings:

- Read all the messages and all the users from the database
- Select the user that match the username provided as argument and obtain its last read id
- Select the subset of messages that have an id greater than the last read id
- Call updateLastRead with the id of the message that has the greatest id
- Return the subset of messages

updateLastReadId should do the followings:

- Read all the users from the database
- Update the last read id of the user matching the username provided as argument with the id provided as argument
- Delete the user database
- Save again all the user previously read

Note that updating the last read id of a user like this is highly inefficient as it requires to read all the messages once , all the users twice, and rewrite the user database file. Similarly, reading the most N recent messages as we implemented in ChatSys 0.1 is also rather inefficient as it also requires to read all the messages once. In a real world scenario with a lot of users and a large database, this would be a serious issue. However, don’t worry about it for the moment. Later, we will replace our dummy text database with a real SQL database that will be far more efficient.

Don’t forget to write unit tests!

Finally, create a CLI similarly as we did before by creating a new class in programming3.chatsys.cli and update the version of ChatSys in the pom.xml. Commit your changes and create a git tag “0.3”.