James Witts S14139866

Advanced programming CMP5308

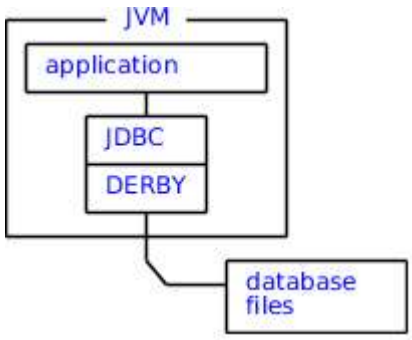
Introduction

This folder contains:

1. All source files necessary to run the Text Message Server (TMS.)
   - Location: `/JamesWitts_s14139866_TMS/Distribution – TMS/msgServer`
2. An *Embedded Derby* database
   - Location: `/JamesWitts_s14139866_TMS/Distribution – TMS/s14139866db`
3. A copy of Apache Derby, an open source relational database implemented entirely in Java and available under the Apache License, Version 2.0.
   - Location: `/JamesWitts_s14139866_TMS/db-derby-10.13.1.1-bin`
   - Driver: `/JamesWitts_s14139866_TMS/db-derby-10.13.1.1-bin/lib/derby`

(Run the `/JamesWitts_s14139866_TMS/Distribution – TMS` as the eclipse project folder.)

The message server creates a connection to the database running in the same environment. This means that a java application being run in another environment while this one is active cannot connect to the database:

> When an application accesses a Derby database using the Embedded Derby JDBC driver, the Derby engine does not run in a separate process, and there are no separate database processes to start up and shut down. Instead, the Derby database engine runs inside the same Java Virtual Machine (JVM) as the application. So, Derby becomes part of the application just like any other jar file that the application uses. Figure 1 depicts this embedded architecture.
>
> **Figure 1: Derby Embedded Architecture**
>
> 
>
>
> The Apache DB Project (2017) [i]

User authentication was already implemented for the TMS system where the user must provide a valid name and password. User authentication is stored in the `userAuth.properties` file is for use with the `MsgSvrConnection` class.

Future development could include providing an improved security policy with the system and the database. Derby has it's own authentication and authorization options where it grants can grant user access to the Derby system but not necessarily access to the database made in the connection request (e.g. for *Create, Read, Update and Delete (CRUD)* operations). This could be implemented by creating the jdbc connection

object in each instance of the `MsgSvrConnection` class where the relevant authenticated properties object can be passed into the constructor for authorization.

Currently the password is also stored in the database for demonstration purposes only.

Testing

Testing the TMS system is done by running the `MessageServer.java` file to create a server at `localhost:9801` on the local machine and then using a telnet client (for example on *Windows 7*, from the command line) type:

```
telnet
```

```
set localecho
```

```
open localhost 9801
```

During telnet sessions please refer to the `userAuth.properties` file to authenticate users against entries in the file. The `userAuth.properties` has four users that can be used for testing purposes:

User	Password
j0	0
j1	1
j2	2
cd	0

Update log

----------Command.java----------

Now can throw an SQL Exception.

----------CommandFactory.java----------

1.0.0.3) Accepts a “Connection jdbc” object into the getCommand()

```
public static msgServer.Command getCommand(BufferedReader in,
BufferedWriter out,
     MsgSvrConnection serverConn, Connection jdbc) throws IOException{
```

1.0.0.0) Added a text menu of commands for the client.

```
out.write("101 LOGIN:\r\n");
```

```
...
```

```
out.write("Enter a command: ");
```

```
out.flush();
```

1.0.0.1) Passed in “jdbc” object to LoginCommand().

```
case MsgProtocol.LOGIN: // 101
     return new LoginCommand(in, out, serverConn, jdbc);
```

1.0.0.1) Added a jdbc to LoginCommand call.

----------EchoClient.java----------

Work in progress, use an alternative client.

----------GetAllMsgsCommand.java----------

This has been implemented using the step-by-step instructions.

----------JDBC_CreateDB.java----------

This was used to create my database and its initial users.

----------JDBC_InsertUserInfo.java----------

This class is used by the `RegisterUser` class and is kept separate because it works on the “jdbc” connection object.

----------JDBC_QueryDB.java----------

This class is used by multiple classes so is kept separate because it can query the `userInfo` table in the database.

----------JDBC_UpdateUserInfo.java----------

This class is used by the `RegisterUserUpdate` class and is kept separate because it works on the “jdbc” connection object.

----------LoginCommand.java----------

Accepts a jdbc connection object to query all the users and their login details

----------MessageServer.java----------

1.0.0.0) Added two string attributes to the class, jdbc_driver and jdbc_url.

```
private final String JDBC_DRIVER =
"org.apache.derby.jdbc.EmbeddedDriver";
```

```
private final String JDBC_URL = "jdbc:derby:s14139866db;create=true";
```

1.0.0.1) Added new connection from server to database in startService().

```
Class.forName(JDBC_DRIVER).newInstance();
```

```
Connection jdbc = DriverManager.getConnection(JDBC_URL);
```

1.0.0.2) Added two methods for storing information to the properties file and object.

```
public void storeUserInfo(String usr, String pass) throws IOException{
      userInfo.setProperty(usr,pass);
      FileOutputStream fout = new
FileOutputStream(MsgProtocol.PASSWORD_FILE);
      userInfo.store(fout,usr+pass);
      fout.close();
}
public void removeUserInfo(String usr) throws IOException{
      userInfo.remove(usr);
      FileOutputStream fout = new
FileOutputStream(MsgProtocol.PASSWORD_FILE);
      userInfo.store(fout,null);
```

```
      fout.close();
}
```

----------MsgProtocol.java----------

1.0.0.0) Change to how the path string is made to the “userAuth.properties” file.

```
public static final String BASE_PATH = new File("").getAbsolutePath();
public static final String PASSWORD_FILE = BASE_PATH.concat("\\
userAuth.properties");
```

1.0.0.1) Added more protocols

```
public static final int GET_ALL_MESSAGES = 106;
[...]
public static final int REMINDER_UPDATES = 110;
```

----------MsgSvrConnection.java----------

1.0.0.0) Added the jdbc connection as an attribute to the class

1.0.0.1) Added the jdbc connection into the constructor

1.0.0.2) Passed the jdbc into the getCommand()

```
command = CommandFactory.getCommand(reader, writer, this, jdbc);
```

----------RegisterUser.java----------

New class

Testing-results

Tests have shown that the `in.readline()` method will insert a backspace as a character if the user tries to edit their entry on `RegisterUser`

----------RegisterUserUpdate.java----------

New class

----------Reminders.java----------

Work in progress

----------ReminderUpdate.java----------

Work in progress

i ‘Embedded Derby’, The Apache DB project
<https://db.apache.org/derby/papers/DerbyTut/embedded_intro.html> accessed 12/06/2017