1. When you are writing code, it’s important that everyone can understand your code, in the case of this code it’s important to keep in mind that every variable you have declared in the code must be used in it, in this case the variable ***initialized***is never used. On the other hand, In the definition of the *Map* Object it’s not clear what kind of values it has, For that reason it’s going to be better if you make the definition as a Map <k,v> where the type of the key and value depends on the application, for this case both values are String because the definition of a ***Properties*** object, the key and the value must be Strings. In order to make your code easier to understand for other people it’s important to use good practices when you are programming, for example the variable’s names should always starts with a lower case letter, use understandable names for variable and methods, for example in the given code there were variables with names like ‘t’ or ‘l’ because it’s hard to understand what is it used for.

When we are using connections with a database it’s important to always manage the possible exceptions that the connection may throw, because any unhandled exception can cause failures in the execution of your code. Finally, it’s important to always close the connection and the statements when the execution of the code ends in order to release de connection and avoid possible errors in the database.

After applying the feedback provided the code looks as below

import java.io.File;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.Statement;

import java.text.DateFormat;

import java.util.Date;

import java.util.Map;

import java.util.Properties;

import java.util.logging.ConsoleHandler;

import java.util.logging.FileHandler;

import java.util.logging.Level;

import java.util.logging.Logger;

public class JobLogger {

private static boolean logToFile;

private static boolean logToConsole;

private static boolean logMessage;

private static boolean logWarning;

private static boolean logError;

private static boolean logToDatabase;

private static Map<String, String> dbParams;

private static Logger logger;

public JobLogger() {

}

public JobLogger(boolean logToFileParam, boolean logToConsoleParam,

boolean logToDatabaseParam, boolean logMessageParam,

boolean logWarningParam, boolean logErrorParam,

Map<String, String> dbParamsMap) {

logger = Logger.getLogger("MyLog");

logError = logErrorParam;

logMessage = logMessageParam;

logWarning = logWarningParam;

logToDatabase = logToDatabaseParam;

logToFile = logToFileParam;

logToConsole = logToConsoleParam;

dbParams = dbParamsMap;

}

public static void LogMessage(String messageText, boolean message,

boolean warning, boolean error) throws Exception {

messageText.trim();

if (messageText == null || messageText.length() == 0) {

return;

}

if (!logToConsole && !logToFile && !logToDatabase) {

throw new Exception("Invalid configuration");

}

if ((!logError && !logMessage && !logWarning)

|| (!message && !warning && !error)) {

throw new Exception("Error or Warning or Message must be specified");

}

Connection connection = null;

Properties connectionProps = new Properties();

connectionProps.put("user", dbParams.get("userName"));// String

connectionProps.put("password", dbParams.get("password")); // String

try {

connection = DriverManager

.getConnection(

"jdbc:" + dbParams.get("dbms") + "://"

+ dbParams.get("serverName") + ":"

+ dbParams.get("portNumber") + "/",

connectionProps);// SQL connection

int type = 0;

if (message && logMessage) {

type = 1;

}

if (error && logError) {

type = 2;

}

if (warning && logWarning) {

type = 3;

}

Statement stmt = connection.createStatement();

try {

String logMessageText = null;

File logFile = new File(dbParams.get("logFileFolder")

+ "/logFile.txt");

if (!logFile.exists()) {

logFile.createNewFile();

}

FileHandler fh = new FileHandler(dbParams.get("logFileFolder")

+ "/logFile.txt");

ConsoleHandler ch = new ConsoleHandler();

if (error && logError) {

logMessageText = logMessageText

+ "error "

+ DateFormat.getDateInstance(DateFormat.LONG)

.format(new Date()) + messageText;

}

if (warning && logWarning) {

logMessageText = logMessageText

+ "warning "

+ DateFormat.getDateInstance(DateFormat.LONG)

.format(new Date()) + messageText;

}

if (message && logMessage) {

logMessageText = logMessageText

+ "message "

+ DateFormat.getDateInstance(DateFormat.LONG)

.format(new Date()) + messageText;

}

if (logToFile) {

logger.addHandler(fh);

logger.log(Level.INFO, messageText);

}

if (logToConsole) {

logger.addHandler(ch);

logger.log(Level.INFO, messageText);

}

if (logToDatabase) {

stmt.executeUpdate("insert into Log\_Values('" + message

+ "', " + String.valueOf(type) + ")");

}

} finally {

stmt.close();

}

} finally {

connection.close();

}

}

}

1. After applying the changes of points 1-5 we have the code below. It was added a private method known as JobLogger (Constructor), in this method it has ridden the file config.properties it contains the information it had the same parameters that the Map Object ***dbParams*** . As well, in this method it was added the connection using the singleton Pattern that allows to open only one instance of the object and it’s possible to used it in anywhere need it. After implementing that, it is only necessary to call the instance and get the connection in order to create the statement. As we know we have the same pattern about closing all the statements and connections after finishing using it. Other method was created for instancing the object that was created. Finally, it was created a method for closing the connection.

import java.io.File;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.io.InputStream;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.sql.Statement;

import java.text.DateFormat;

import java.util.Date;

import java.util.Properties;

import java.util.logging.ConsoleHandler;

import java.util.logging.FileHandler;

import java.util.logging.Level;

import java.util.logging.Logger;

public class JobLogger {

private static boolean logToFile;

private static boolean logToConsole;

private static boolean logMessage;

private static boolean logWarning;

private static boolean logError;

private static boolean logToDatabase;

private static Logger logger;

private static JobLogger jobLog;

private static Connection connection;

private static Properties connectionProps;;

private JobLogger() throws SQLException {

connectionProps = new Properties();

try {

InputStream in = new FileInputStream("config.properties");

connectionProps.load(in);

in.close();

connection = DriverManager.getConnection(

"jdbc:" + connectionProps.get("dbms") + "://"

+ connectionProps.get("serverName") + ":"

+ connectionProps.get("portNumber") + "/",

connectionProps.getProperty("userName"),

connectionProps.getProperty("password"));

} catch (FileNotFoundException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

}

public Connection getConnection() {

return connection;

}

public JobLogger getInstance() throws SQLException {

if (jobLog == null)

jobLog = new JobLogger();

return jobLog;

}

public static void closeConnection() throws SQLException {

if (connection != null)

connection.close();

}

public JobLogger(boolean logToFileParam, boolean logToConsoleParam,

boolean logToDatabaseParam, boolean logMessageParam,

boolean logWarningParam, boolean logErrorParam) {

logger = Logger.getLogger("MyLog");

logError = logErrorParam;

logMessage = logMessageParam;

logWarning = logWarningParam;

logToDatabase = logToDatabaseParam;

logToFile = logToFileParam;

logToConsole = logToConsoleParam;

}

public static void LogMessage(String messageText, boolean message,

boolean warning, boolean error) throws Exception {

messageText.trim();

if (messageText == null || messageText.length() == 0) {

return;

}

if (!logToConsole && !logToFile && !logToDatabase) {

throw new Exception("Invalid configuration");

}

if ((!logError && !logMessage && !logWarning)

|| (!message && !warning && !error)) {

throw new Exception("Error or Warning or Message must be specified");

}

int type = 0;

if (message && logMessage) {

type = 1;

}

if (error && logError) {

type = 2;

}

if (warning && logWarning) {

type = 3;

}

JobLogger instance = jobLog.getInstance();

try {

Statement stmt = instance.getConnection().createStatement();

try {

String logMessageText = null;

File logFile = new File(connectionProps.get("logFileFolder")

+ "/logFile.txt");

if (!logFile.exists()) {

logFile.createNewFile();

}

FileHandler fh = new FileHandler(

connectionProps.get("logFileFolder") + "/logFile.txt");

ConsoleHandler ch = new ConsoleHandler();

if (error && logError) {

logMessageText = logMessageText

+ "error "

+ DateFormat.getDateInstance(DateFormat.LONG)

.format(new Date()) + messageText;

}

if (warning && logWarning) {

logMessageText = logMessageText

+ "warning "

+ DateFormat.getDateInstance(DateFormat.LONG)

.format(new Date()) + messageText;

}

if (message && logMessage) {

logMessageText = logMessageText

+ "message "

+ DateFormat.getDateInstance(DateFormat.LONG)

.format(new Date()) + messageText;

}

if (logToFile) {

logger.addHandler(fh);

logger.log(Level.INFO, messageText);

}

if (logToConsole) {

logger.addHandler(ch);

logger.log(Level.INFO, messageText);

}

if (logToDatabase) {

stmt.executeUpdate("insert into Log\_Values('" + message

+ "', " + String.valueOf(type) + ")");

}

} finally {

stmt.close();

}

} catch (SQLException e) {

e.printStackTrace();

} finally {

JobLogger.closeConnection();

}

}

}