Code Review / Refactoring exercise

Please review the following code snippet. Assume that all referenced assemblies have been properly included.

The code is used to log different messages throughout an application. We want the ability to be able to log to a text file, the console and/or the database. Messages can be marked as message, warning or error. We also want the ability to selectively be able to choose what gets logged, such as to be able to log only errors or only errors and warnings.

1. **If you were to review the following code, what feedback would you give? Please be specific and indicate any errors that would occur as well as other best practices and code refactoring that should be done.**

1.1 Comentarios

- Se tienen varias funcionalidades y métodos unidos en una sola clase. Lo que afectaría posteriormente en el mantenimiento del código existente ante una extensión o mejoras en la funcionalidad.

1.2 Errores Posibles

- SQLException
- IOException

1.3 Buenas Practicas

- Variables que no se usen, deben ser eliminadas.
- Si existen métodos genéricos estos deben ser llevados a una clase de tipo Abstract para su implementación en una clase “Hija”. (Por ejemplo: La conexión a base de datos)
- Externalizar en un archivo de propiedades los valores de conexión.
- Hacer uso de los principios de POO: Polimorfismo y Abstracción.
- Documentar todos los métodos y clases.
- Los nombres de las variables y métodos deben ser legibles y/o entendibles.
- Los valores que sean fijos deben ser constantes.
- Los métodos de las interfaces que contenga lógica de negocio deben ser probadas y/o testeadas con todos sus casos de uso.

2. **Rewrite the code based on the feedback you provided in question 1. Please include unit tests on your code.**

```
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 boolean initialized;
	private static Map dbParams;
	private static Logger logger;

	public JobLogger(boolean logToFileParam, boolean logToConsoleParam, boolean logToDatabaseParam,
			boolean logMessageParam, boolean logWarningParam, boolean logErrorParam, Map
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"));
		connectionProps.put("password", dbParams.get("password"));

		connection = DriverManager.getConnection("jdbc:" + dbParams.get("dbms") + "://" +
dbParams.get("serverName")
				+ ":" + dbParams.get("portNumber") + "/", connectionProps);

		int t = 0;
		if (message && logMessage) {
			t = 1;
		}

		if (error && logError) {
			t = 2;
		}
```

```
                if (warning && logWarning) {
                        t = 3;
                }

                Statement stmt = connection.createStatement();

                String l = 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) {
                        l = l + "error " + DateFormat.getDateInstance(DateFormat.LONG).format(new Date()) +
messageText;
                }

                if (warning && logWarning) {
                        l = l + "warning " +DateFormat.getDateInstance(DateFormat.LONG).format(new Date()) +
messageText;
                }

                if (message && logMessage) {
                        l = l + "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(t) + ")");
                }
        }
}
```

Class Refactoring

```
package com.belatrix.tech.task.data.reference;

import com.belatrix.tech.task.data.creating.CreateLogService;
import com.belatrix.tech.task.data.logging.LoggingConsole;
import com.belatrix.tech.task.data.logging.LoggingDataBase;
```

```java
import com.belatrix.tech.task.data.logging.LoggingFile;
import com.belatrix.tech.task.data.logging.LogsManagement;
import com.belatrix.tech.task.data.validate.ValidateEntryService;

import java.util.Map;
import java.util.logging.Logger;

/**
 * Class implement functions of:
 *  # Initializer constructor variables.
 *  # Refactoring method void logMessage():
 *      # Validate inputs
 *      # Create message
 *      # Registry message
 */
public class JobLoggerFinal {

 private static boolean logToFile;
 private static boolean logToConsole;
 private static boolean logToDatabase;

 private static boolean logMessage;
 private static boolean logWarning;
 private static boolean logError;


 //private boolean initialized; Eliminar codigo muerto, para un tener un codigo
limpio.

 private static Map dbParams;
 private static Logger logger;

 private CreateLogService logService;
 private ValidateEntryService entryService;

 public JobLoggerFinal(boolean logToFileParam, boolean logToConsoleParam,
boolean logToDatabaseParam,
            boolean logMessageParam, boolean logWarningParam, boolean
logErrorParam, Map parameters,
            CreateLogService logService, ValidateEntryService entryService){
  logger = Logger.getLogger("NyLog");
  logError = logErrorParam;
  logMessage = logMessageParam;
  logWarning = logWarningParam;
  logToDatabase = logToDatabaseParam;
  logToFile = logToFileParam;
  logToConsole = logToConsoleParam;
  dbParams = parameters;

  this.logService = logService;
  this.entryService = entryService;
 }

 public void logMessage(String message) throws Exception {

   entryService.validateValues(message, logToConsole, logToFile,
     logToDatabase, logMessage, logError, logWarning);
   message = message.trim();

   String newMessageLog = logService.createMessageLog(message, logError,
logWarning, logMessage);

   if(logToFile){
    LogsManagement log = new LoggingFile(newMessageLog, dbParams, logger);
```

```
     log.insertLogs();
    }

    if(logToConsole){
     LogsManagement log = new LoggingConsole(message, logger);
     log.insertLogs();
    }

    if(logToDatabase){
     int typeLog = logService.getSelectTypeLog(logError, logWarning, logMessage);
     LogsManagement log = new LoggingDataBase(message, typeLog, dbParams);
     log.insertLogs();
    }



 }

}
```

Responsibility: Denis Raul Choquehuanca Vargas