UNIVERSIDAD DE LAS FUERZAS ARMADAS – ESPE

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

OBJECT ORIENTED PROGRAMMING

Student: Llumiquinga Moreno Jerson.

E-mail: jsllumiquinga1@espe.edu.ec

#Cell: 0980460057

Professor: Edison Lascano.

E-mail: jelascano@espe.edu.ec

#Cell: 0961195050

NRC: 14539

MAY24 - SEP24

Handling null in birthDate:

Problem: If the calculateAge method receives a birthDate that is null, a NullPointerException will be thrown when trying to set the time to birthCalendar.

Solution: Add a check at the beginning of the method to handle this case.

```java
package ec.edu.espe.academygradesystemfrm.controller;

import java.util.Calendar;
import java.util.Date;
/**
 *
 * @author Jerson Llumiquinga M. - TEAM: JEZHEA S.E.A
 */
public class AgeCalculator {
public static int calculateAge(Date birthDate) {
        Calendar birthCalendar = Calendar.getInstance();
        birthCalendar.setTime(birthDate);
        Calendar today = Calendar.getInstance();

        int age = today.get(Calendar.YEAR) - birthCalendar.get(Calendar.YEAR);
        if (today.get(Calendar.DAY_OF_YEAR) < birthCalendar.get(Calendar.DAY_OF_YEAR)) {
            age--;
        }
        return age;
    }
}
```

Close the connection after each insert:

Problem: In the insertProfessor method, the connection to MongoDB is closed right after inserting a teacher. If insertProfessor is called multiple times, it will open and close the connection on each call, which could be inefficient.

Solution: Keep the connection open while the application is running and close it only when it is no longer needed, such as when ending the application.

```java
package ec.edu.espe.academygradesystemfrm.controller;

import com.mongodb.MongoException;
import com.mongodb.client.MongoCollection;
import ec.edu.espe.academygradesystemfrm.model.CreateProfessor;
import ec.edu.espe.academygradesystemfrm.utils.ProfessorToMongo;
import org.bson.Document;

/**
 *
 * @author Lainez Ricardo JEZHE SEA - ESPE
 */
public class CreateProfessorController {
    private ProfessorToMongo mongoDBConnection;

    public CreateProfessorController(){
        mongoDBConnection = new ProfessorToMongo();
    }

    public void insertProfessor(CreateProfessor professor){
        MongoCollection<Document> collection = mongoDBConnection.getCollection("professors");
        try{
            collection.insertOne(professor.toDocument());
            mongoDBConnection.closeConnection();
        }catch(MongoException e){
            e.printStackTrace();
        }
```

Close the connection after each insert:

Problem: In the insertProfessor method, the connection to MongoDB is closed right after inserting a teacher. If insertProfessor is called multiple times, it will open and close the connection on each call, which could be inefficient.

Solution: Keep the connection open while the application is running and close it only when it is no longer needed, such as when ending the application.

```java
package ec.edu.espe.academygradesystemfrm.controller;

import java.awt.Color;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JTextField;
/**
 *
 * @author Jerson Llumiquinga M. - TEAM: JEZHEA S.E.A
 */
public class GradeCalculator {
    public static double calculateAverage(double firstTerm, double secondTerm, double thirdTerm) {
        return (firstTerm + secondTerm + thirdTerm) / 3;
    }

    public static String determineStatus(double average, JLabel statusLabel) {
        if (average >= 14) {
            statusLabel.setText("Aprobado");
            statusLabel.setForeground(Color.GREEN);
            return "Aprobado";
        } else {
            statusLabel.setText("Desaprobado");
            statusLabel.setForeground(Color.RED);
            return "Desaprobado";
        }
    }

    public static boolean validateGrades(JTextField... gradeFields) {
```

```java
        } else {
            statusLabel.setText("Desaprobado");
            statusLabel.setForeground(Color.RED);
            return "Desaprobado";
        }
    }

    public static boolean validateGrades(JTextField... gradeFields) {
        try {
            for (JTextField field : gradeFields) {
                Double.parseDouble(field.getText());
            }
            return true;
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(null, "Por favor, ingrese valores numéricos válidos en los parciales."
            return false;
        }
    }
}
```

Problem:

Previously, there was duplicate code when validating data. Both in the lD entry of the teacher and the student.

Solution:

We can create a class that validates the fields, since both inputs require the same data types.

```java
package ec.edu.espe.academygradesystemfrm.controller;

import java.awt.Color;
import javax.swing.JOptionPane;
import javax.swing.JTextField;
import javax.swing.JComboBox;
/**
 *
 * @author Jerson Llumiquinga M. - TEAM: JEZHEA S.E.A
 */

public class ValidateData {
      public static boolean validateIdLength(String idText, JTextField textField) {
        if (idText.length() > 10) {
            JOptionPane.showMessageDialog(null, "El ID no debe tener más de 10 dígitos.", "Error de entrada", JO
            textField.setForeground(Color.RED);
            textField.requestFocus();
            return false;
        }
        return true;
    }

    public static boolean validateIdIsInteger(String idText, JTextField textField) {
        try {
            Integer.parseInt(idText);
            textField.setForeground(Color.BLACK);
            return true;
        } catch (NumberFormatException e) {
            JOptionPane.showMessageDialog(null, "El ID debe ser un número entero.", "Error de entrada", JOptionP
            textField.setForeground(Color.RED);
            textField.setText("");
```

Repetitive Creation and Closing of MongoClient:

Problem: The createMongoClient method is called repeatedly, creating a new MongoClient connection each time the database needs to be accessed. This can be inefficient and could lead to performance issues or connection limits.

Solution: Consider creating a single MongoClient instance and reusing it as long as the application is active.

```java
import ec.edu.espe.academygradesystemfrm.model.CreateStudent;
import org.bson.Document;

/**
 *
 * @author Lainez Ricardo JEZHE SEA - ESPE
 */

public class StudentToMongo {
    private static MongoClient createMongoClient() {
    String connectionString = "mongodb+srv://jezhe:jezheoop@cluster0.6vuzzwl.mongodb.net/";
        ServerApi serverApi = ServerApi.builder().version(ServerApiVersion.V1).build();

        MongoClientSettings settings = MongoClientSettings.builder()
                .applyConnectionString(new ConnectionString(connectionString)).serverApi(serverApi).build();
        return MongoClients.create(settings);
    }
    public static CreateStudent getStudentById(int studentId) {
    CreateStudent student = null;

    try (MongoClient mongoClient = createMongoClient()) {
        MongoDatabase database = mongoClient.getDatabase("AcademyGradeRegister");
        MongoCollection<Document> collection = database.getCollection("students");

        Document query = new Document("id", studentId);
        Document studentDocument = collection.find(query).first();
```

Lack of Proper Exception Handling:

Problem: Using e.printStackTrace() is useful for debugging but is not suitable for production environments. Additionally, generic exceptions are being caught, which can make it difficult to identify specific problems.

Solution: Implement more detailed exception handling and use a logging framework like SLF4J to log errors.

```java
        return student;
    }

    public static void uploadSudentData(CreateStudent student){
        try(MongoClient mongoClient = createMongoClient()){
            MongoDatabase database = mongoClient.getDatabase("AcademyGradeRegister");

            saveStudentToDatabase(student, database);

        }catch(Exception e){
            e.printStackTrace();
        }
    }

    private static void saveStudentToDatabase(CreateStudent student, MongoDatabase database){
        MongoCollection<Document> collection = database.getCollection("students");
        Document studentDocument = new Document("id", student.getId())
                .append("nombre", student.getName())
                .append("grado", student.getDegree())
                .append("edad", student.getAge());

        try{
            collection.insertOne(studentDocument);
            System.out.println("student guardado exitosamente!!");
        }catch(MongoException e){
            e.printStackTrace();
        }
```