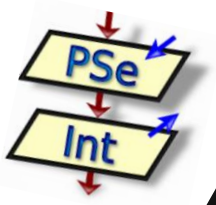
UNIVERSIDAD DE LAS FUERZAS ARMADAS – ESPE

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**

**Single Responsibility Principle (SRP):**

This principle establishes that a class must have a single reason for changing, that is, it must only have one responsibility. In our code, the StudentToMongo class handles both the connection to MongoDB and the business logic related to obtaining and storing students. This can be considered a violation of the SRP, since the class is handling two different responsibilities: database connection and business logic.

**Solution: We can separate the responsibilities by creating two classes: one for the connection to MongoDB and another for the business logic. For example:**

**Open/Closed Principle (OCP):**

This principle states that classes should be open for extension, but closed for modification. In our case, if you needed to change the way a student is saved or retrieved from the database, you would need to modify StudentToMongo. This suggests a lack of adherence to the OCP.

**Solution: We could use interfaces to define database operations, allowing different implementations. For example, a StudentRepositoryInterface interface with getStudentById and saveStudent methods, and then implement this interface in the StudentRepository class. This would make it easier to extend the functionality without modifying the existing class.**

**Dependency Inversion Principle (DIP):**

This principle states that high-level modules should not depend on low-level modules; both must depend on abstractions. Currently, StudentToMongo depends directly on the MongoDB implementation, making it difficult to change the database or storage form without modifying the code.

**Solution: We can use dependency injection to pass an implementation of StudentRepositoryInterface to the class that needs it, instead of instantiating MongoClient directly within the class. This would allow the class to depend on abstractions rather than concrete implementations.**

**Single Responsibility Principle (SRP):**

Detected Issue: The GradeCalculator class currently has multiple responsibilities:

Calculate the average of the grades.

Determine the status (approved/failed) and update a graphical interface component (JLabel).

Validate the notes entered in the text fields of the graphical interface (JTextField).

This violates SRP because the class mixes business logic (average and state calculation) with presentation logic (GUI component manipulation and input validation).

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
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";
        }
    }
```

```java
    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;
        }
    }
}
```

**Solution: Separate these responsibilities into different classes:**

**GradeStatusUpdater Class: Responsible for updating the graphical interface with the approval status.**

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package ec.edu.espe.strategygui.view;

/**
 *
 * @author Jerson Llumiquinga M. - TEAM: JEZHEA S.E.A
 */
public class GradeStatusUpdater {
    public class GradeStatusUpdater {
    public static String updateStatus(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";
        }
    }
}
}
```

**GradeValidator Class: Responsible for validating the entered grades**

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package ec.edu.espe.strategygui.view;

/**
 *
 * @author Jerson Llumiquinga M. - TEAM: JEZHEA S.E.A
 */
public class GradeValidator {
    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;
        }
    }
}
```

**Open/Closed Principle (OCP):**

The determineStatus and validateGrades logic is coupled to the graphical interface. If you wanted to change how the interface is handled or add new validation criteria, you would have to modify the GradeCalculator class.

**Solution: Implement interfaces or use design patterns that allow you to extend the functionality without modifying the existing class. For example, you could use a strategy pattern to handle different validation logic or determine states.**

**Dependency Inversion Principle (DIP):**

The GradeCalculator class depends directly on specific GUI components (JLabel, JTextField, JOptionPane), making it difficult to reuse the calculation and validation logic in a non-graphical environment or with different interfaces.

Use dependency injection to pass interfaces to GUI components or separate business logic from presentation logic. For example, GradeStatusUpdater could accept an interface that encapsulates the setStatus operation instead of a specific component like JLabel.