Pregunta Ejemplo Control 1

IIC2113-1 2022

Profesor Antonio Ossa

Considere el siguiente código en Ruby, el cual es parte de un programa de gestión de estudiantes y cursos a nivel universitario. En base a lo anterior, responda las siguientes preguntas.

```
class DCCCourse < UniversityCourse

    def initialize(acronym, name, instructor_name)
        # Course acronym (ex. "IIC2113")
        @acronym = acronym
        # Course name (ex. "Diseño Detallado de Software")
        @name = name
        # Course instructor (ex. "Antonio Ossa")
        @instructor = Instructor.new(instructor_name)
        # Number of students
        @n_students = 0
        # Keep students grades apart from the students themselves
        @students = Hash.new
        @students_grades = Hash.new
    end

    def create_course_calendar
        # Use superclass method instead of overriding
        super
    end

    def create_field_trip
        # DCC courses do not implement field trips :(
        raise NotImplementedError, "there are no field trips at DCC"
    end

    def assign_classroom(classroom)
        # Assign classroom when university decides which one
        @classroom = Classroom.new(classroom)
    end

    def add_student(student_full_name, student_email)
        # Create a student instance
        student = Student.new(student_full_name, student_email)
        # Store instance in students hash
        @student[student.unique_id] = student
        # Update students counter
        @n_students += 1
    end

    def get_student(student_full_name, student_email)
        # Create a fake student instance
        student = Student.new(student_full_name, student_email)
        # Query students hash using fake student instance to
```

```
        # retrieve original student instance
        @student[student.unique_id]
    end

    def add_participation_bonus(student_full_name, student_email)
        # Create a fake student instance
        student = Student.new(student_full_name, student_email)
        # Query students hash using fake student instance to
        # retrieve original student grades
        @student_grades[student.unique_id] += 0.1
    end

    def get_student_grades(student_full_name, student_email)
        # Create a fake student instance
        student = Student.new(student_full_name, student_email)
        # Query students hash using fake student instance to
        # retrieve original student grades
        @student_grades[student.unique_id]
    end

    def publish_assignment(assignment)
        # Depending on the assignment class
        # publish using the instance methods
        case assignment
        when Homework
            assignment.submit_to_canvas
        when Project
            if assignment.requires_groups
                assignment.create_groups
            end
            assignment.submit_to_github
        when Interrogation
            assignment.send_to_printer
            assignment.start_test
        end
    end

    def export_grades_as_json
        # Export students grades as a JSON file
        File.open("#{@acronym}.json") { |file|
            file.puts JSON.pretty_generate(@student_grades)
        }
    end

    def export_grades_as_yaml
        # Export students grades as a YAML file
        File.open("#{@acronym}.json") { |file|
            file.write @student_grades.to_yaml
        }
    end

end
```

1. Identifique el o los principios SOLID que se transgreden en el código y explique por qué brevemente.

1. **SRP**: El principio de responsabilidad única se transgrede pues esta clase no solo representa y modela a un curso del DCC, sino que también se encarga de exportar las notas a un formato específico. Podría solucionarse con una clase para exportar instancias a cada tipo.
2. **OCP**: El principio de abierto/cerrado se transgrede pues si se agrega un nuevo tipo de evaluación (assignment), necesariamente tendremos que modificar esta clase para soportarlo (método publish_assignment). Podría solucionarse si existiera un método publish en todas las subclases de Assignment
3. **LSP**: El principio de substitución de Liskov se rompe puesta las instancias de la clase DCCCourse no podrán reemplazar a las de UnviersityCourse, en particular a aquellas que utilicen el método create_field_trip, pues en esta clase no está implementado y lanzará una excepción si se utiliza. Podría solucionarse con un mejor modelamiento, no obligando a todos los cursos a implementar el método.
4. **ISP**: Este principio no se transgrede de manera obvia. Si existiera un método que fácilmente pueda o deba ser dividido en dos, este principio se rompería.
5. **DIP**: El principio de inversión de dependencia se rompe pues en varias partes inicializamos instancias en lugar de recibir instancias desde fuera de la clase (Instructor, Classroom y Student). Para invertir la dependencia tendríamos que recibir instancias de esas clases como argumentos en cada método.

2. Identifique el o los code smells presentes en el siguiente código y explique por qué brevemente.

Bloaters:

- *Long Method*: No aplica.
- *Large Class*: Sí aplica. Hay muchos métodos
- *Primitive Obsession*: Sí aplica. En lugar de ocupar instancias de objetos en muchos lugares ocupamos variables (métodos relacionados con student y uso de n_students)
- *Long Parameter List*: No aplica
- *Data Clumps*: No aplica.

Object-Orientation Abusers:

- *Alternative Classes with Different Interfaces*: No aplica, solo vemos una clase
- *Refused Bequest*: Sí aplica. No estamos utilizando el método create_field_trip, entonces no hay necesidad real de heredarlo
- *Switch Statements*: Sí aplica, es evidente en el método publish_assignment
- *Temporary Field*: No aplica, aunque si bien podría justificarse con el método assign_classroom no hay evidencia de que este se elimine y/o sea utilizado solo en ciertas circunstancias

Change Preventers:

- *Divergent change*: No aplica, no hay evidencia suficiente
- *Parallel Inheritance Hierarchies*: No aplica, no hay evidencia suficiente

- *Shotgun Surgery*: No aplica, pero podría ser justificado por medio de la clase Student (o incluso Instructor y Classroom), ya que un cambio en su inicializador implica tener que hacer cambios en esta clase

Dispensables:

- *Comments*: Sí aplica. A nivel de clase prácticamente
- *Duplicate Code*: Sí aplica. Nuevamente, muchas veces inicializamos Student de la misma manera
- *Data Class*: No aplica
- *Lazy Class*: No aplica
- *Speculative Generality*: No aplica.