Algoritmos y Estructuras de Datos II

Trabajo Práctico N° 5	Unidad 5
Modalidad: Semi -Presencial	**Estratégica Didáctica:** Trabajo individual
Metodología de Desarrollo: Det. docente	**Metodología de Corrección:** Via Classroom.
Carácter de Trabajo: Obligatorio – Con Nota	**Fecha Entrega:** A confirmar por el Docente.

MARCO TEÓRICO (Cuestionario reflexivo):

Unidad 4 - Trabajo Práctico – GRASP Responsabilidad en Diseño Orientado a Objetos.

Responder el siguiente cuestionario en función de la bibliografía obligatoria.

1. Comentar y Reflexionar sobre las citas de Craig Larman y martin Fowler.

Las citas destacan la importancia de los fundamentos del diseño por encima de las herramientas.

Craig Larman: Afirma que la herramienta de diseño crítica no es UML ni ninguna otra tecnología, sino "una mente bien educada en principios de diseño" . Esto implica que saber dibujar diagramas (UML) no sirve de nada si no se saben aplicar los principios correctos para estructurar el software.

Martin Fowler: Señala que "Entender las responsabilidades es clave para un buen diseño orientado a objetos" . Esto refuerza la idea de que el diseño no se trata solo de clases y métodos, sino de asignar correctamente qué debe hacer cada parte del sistema.

2. Relacionar UML, practicas y Patrones

Se establece una relación jerárquica y funcional entre estos conceptos:

UML: Es simplemente un lenguaje estándar de modelado visual. Conocer sus detalles (la notación) no te enseña a diseñar, de la misma manera que saber gramática no te convierte en un novelista.

Patrones y Prácticas: Son el conocimiento real de diseño. Aprender a diseñar (OOAD) significa dominar los "principios de diseño" y los "patrones de diseño" (como GRASP), no solo la sintaxis de UML .

Relación: Se utiliza UML como una herramienta para visualizar y comunicar esos principios y patrones aplicados.

3. ¿Qué es Responsability Driven Design?

El Diseño Guiado por Responsabilidades (RDD) es un estilo de diseño en el que se piensa en los objetos de software como entidades que tienen responsabilidades, roles y colaboraciones. Es un enfoque donde el diseño se estructura preguntando qué debe hacer cada objeto y qué información debe conocer, en lugar de centrarse solo en los datos.

4. Comentar la cita de Autor “Una responsabilidad no es lo mismo que un método, es una abstracción, pero los métodos cumplen con las responsabilidades.”

Una responsabilidad es una abstracción, un contrato u obligación de un clasificador (clase). Es lo que se espera que haga el objeto desde una perspectiva de diseño.

Los métodos son la implementación concreta. Pueden existir muchos métodos pequeños para cumplir una sola responsabilidad abstracta (como "calcular impuestos"), o una responsabilidad puede implicar la coordinación de varios métodos.

5. ¿Qué entiende por Patrón (en Software y Arquitectura de Software)?

Un patrón es una descripción nombrada de un problema y su solución que se puede aplicar a nuevos contextos. Es una forma de capturar y reutilizar el conocimiento de diseño, proporcionando un esquema probado para resolver problemas recurrentes. En el contexto de GRASP, son principios codificados para asignar responsabilidades

6. ¿Qué significa y qué se pretende con GRASP?

GRASP son las siglas de General Responsibility Assignment Software Patterns (Patrones Generales de Software para la Asignación de Responsabilidades).

Su objetivo es proporcionar guías o principios para ayudar a asignar responsabilidades a clases y objetos de manera sistemática y racional durante el diseño orientado a objetos . Ayudan a entender cómo diseñar objetos esenciales y cómo interactúan

7. Relacionar Clase, Conocer y Hacer.

En RDD y GRASP, las responsabilidades de una Clase (u objeto) se dividen en dos categorías principales que definen su comportamiento:

Hacer (Doing): Implica la acción. Una clase es responsable de hacer algo por sí misma (como calcular algo), iniciar una acción en otros objetos, o controlar y coordinar actividades de otros.

Saber/Conocer (Knowing): Implica la información. Una clase es responsable de conocer sus datos privados (encapsulamiento), conocer los objetos con los que está relacionada, o saber cosas que puede derivar o calcular.

Marco Práctico

1. Usar la Calculadora del Kata que realizamos e Incorporar el Patrón Controlador. Para ello deberá separar la Interfaz de Usuario (consola) con la Lógica que maneje la Ecuación (Suma, Resta, Multiplicación o División)

```h
// ControladorCalculadora.h
#ifndef CONTROLADOR_CALCULADORA_H
#define CONTROLADOR_CALCULADORA_H

#include "Operacion.h"

class ControladorCalculadora {
private:
    Operacion operacion;

public:
    ControladorCalculadora();

    // Método que coordina la operación
    bool procesarOperacion(double op1, double op2, char operador);

    // Obtener el resultado
    double obtenerResultado() const;

    // Validar operador
    bool validarOperador(char operador) const;
};

#endif
```

```cpp
// ControladorCalculadora.cpp
#include "ControladorCalculadora.h"

ControladorCalculadora::ControladorCalculadora() {}

bool ControladorCalculadora::procesarOperacion(double op1, double op2, char operador) {
    // El controlador coordina las operaciones
    operacion.setOperando1(op1);
    operacion.setOperando2(op2);
    operacion.setOperador(operador);

    return operacion.calcular();
}

double ControladorCalculadora::obtenerResultado() const {
    return operacion.getResultado();
}

bool ControladorCalculadora::validarOperador(char operador) const {
    return (operador == '+' || operador == '-' ||
            operador == '*' || operador == '/');
}
```

```h
// InterfazUsuario.h
#ifndef INTERFAZ_USUARIO_H
#define INTERFAZ_USUARIO_H

#include "ControladorCalculadora.h"
#include <string>

class InterfazUsuario {
private:
    ControladorCalculadora* controlador;

public:
    InterfazUsuario(ControladorCalculadora* ctrl);

    // Métodos de entrada/salida
    void mostrarMenu();
    void ejecutar();

private:
    void mostrarResultado(double resultado);
    void mostrarError(const std::string& mensaje);
    double solicitarNumero(const std::string& mensaje);
    char solicitarOperador();
};

#endif
```

```cpp
// InterfazUsuario.cpp
#include "InterfazUsuario.h"
#include <iostream>
using namespace std;

InterfazUsuario::InterfazUsuario(ControladorCalculadora* ctrl)
    : controlador(ctrl) {}

void InterfazUsuario::mostrarMenu() {
    cout << "\n=== CALCULADORA ===" << endl;
    cout << "Operaciones disponibles: +, -, *, /" << endl;
    cout << "====================" << endl;
}

double InterfazUsuario::solicitarNumero(const string& mensaje) {
    double numero;
    cout << mensaje;
    cin >> numero;
    return numero;
}

char InterfazUsuario::solicitarOperador() {
    char operador;
    cout << "Ingrese el operador (+, -, *, /): ";
    cin >> operador;
    return operador;
}

void InterfazUsuario::mostrarResultado(double resultado) {
    cout << "\nResultado: " << resultado << endl;
}

void InterfazUsuario::mostrarError(const string& mensaje) {
    cout << "\nError: " << mensaje << endl;
}

void InterfazUsuario::ejecutar() {
    char continuar;
```

```
    do {
        mostrarMenu();

        double num1 = solicitarNumero("Ingrese el primer número: ");
        char operador = solicitarOperador();

        // Validar operador antes de pedir el segundo número
        if(!controlador->validarOperador(operador)) {
            mostrarError("Operador inválido");
            cout << "\n¿Desea realizar otra operación? (s/n): ";
            cin >> continuar;
            continue;
        }

        double num2 = solicitarNumero("Ingrese el segundo número: ");

        // Procesar a través del controlador
        if(controlador->procesarOperacion(num1, num2, operador)) {
            mostrarResultado(controlador->obtenerResultado());
        } else {
            if(operador == '/' && num2 == 0) {
                mostrarError("No se puede dividir por cero");
            } else {
                mostrarError("Operación inválida");
            }
        }

        cout << "\n¿Desea realizar otra operación? (s/n): ";
        cin >> continuar;

    } while(continuar == 's' || continuar == 'S');

    cout << "\n¡Gracias por usar la calculadora!" << endl;
}
```

```h
// Operacion.h
#ifndef OPERACION_H
#define OPERACION_H

class Operacion {
private:
    double operando1;
    double operando2;
    char operador;
    double resultado;

public:
    Operacion();

    // Setters
    void setOperando1(double op1);
    void setOperando2(double op2);
    void setOperador(char op);

    // Getters
    double getOperando1() const;
    double getOperando2() const;
    char getOperador() const;
    double getResultado() const;

    // Método para calcular
    bool calcular();
};

#endif
```

```cpp
// Operacion.cpp
#include "Operacion.h"

Operacion::Operacion() : operando1(0), operando2(0), operador('+'), resultado(0) {}

void Operacion::setOperando1(double op1) {
    operando1 = op1;
}

void Operacion::setOperando2(double op2) {
    operando2 = op2;
}

void Operacion::setOperador(char op) {
    operador = op;
}

double Operacion::getOperando1() const {
    return operando1;
}

double Operacion::getOperando2() const {
    return operando2;
}

char Operacion::getOperador() const {
    return operador;
}

double Operacion::getResultado() const {
    return resultado;
}
```

```cpp

bool Operacion::calcular() {
    switch(operador) {
        case '+':
            resultado = operando1 + operando2;
            return true;
        case '-':
            resultado = operando1 - operando2;
            return true;
        case '*':
            resultado = operando1 * operando2;
            return true;
        case '/':
            if(operando2 != 0) {
                resultado = operando1 / operando2;
                return true;
            }
            return false; // División por cero
        default:
            return false; // Operador inválido
    }
}
```

```cpp
// main.cpp
#include "InterfazUsuario.h"
#include "ControladorCalculadora.h"

int main() {
    ControladorCalculadora controlador;
    InterfazUsuario interfaz(&controlador);

    interfaz.ejecutar();

    return 0;
}
```

Lic. Oemig José Luis.