

Informe de Deberes

Curso: Programación Orientada a Objetos-1323

Fecha: 08/12/2024

I. Portada

1. **Título del Proyecto:**
Creación de Objetos y UML
 2. **Datos del Estudiante:**
 - Nathaly Stefanía Cusapaz Quiña
 - L00440451
 - nscuasapaz@gmail.com
 3. **Fecha de Entrega:**
 - 08/12/2024
-

II. Índice

1. Introducción
 2. Objetivos del Proyecto
 3. Metodología
 4. Desarrollo del Proyecto
 5. Resultados
 6. Conclusiones
 7. Referencias
 8. Anexos
-

III. Introducción

- **Descripción del Proyecto:**
 - Diseñe 5 objetos diferentes con su correspondiente diagrama UML, asegurándose de mostrar las relaciones entre ellos.
- **Antecedentes:**
 - UML (o Lenguaje Unificado de Modelado, por sus siglas en inglés) es un software de lenguaje de modelado que se usa para representar el diseño de un sistema específico. UML tiene varios tipos de diagramas, que muestran diferentes aspectos de las entidades a representar. También, se puede decir que

el UML es un bosquejo o plano que ayuda al programador a construir un programa, a través de una representación más visual y entendible de las líneas de código (Vélez S., Peña, Gortazar, & Sánchez, 2011)

- Diagrama de clases

Los diagramas de clase son los más utilizados en UML permiten representar un conjunto de clases, interfaces y su relación entre ellos.

Los objetos son entidades que tienen características que los diferencian de otros y realizan diferentes acciones como borrador, lápiz, mesa. En Programación Orientada a Objetos (POO), las características son variables a las que se conoce como atributos y las acciones que realiza un objeto se le conoce como métodos.

Las clases son moldes que permiten crear objetos a partir de atributos y métodos ya establecidos.

- Herencia

La relación entre clases es parte integral de la programación orientada a objetos. Su representación es una línea sólida con una flecha cerrada y hueca, siempre apunta a la clase padre (Codigofacilito, 2010).

- Diagrama de Clases

En la parte superior se coloca los atributos o propiedades, y debajo las operaciones de la clase.

Se inicia con un carácter, esto denotará la visibilidad del atributo o método.

Estos son los niveles de visibilidad:

- private

+ public

protected

~ default

Una forma de representar las relaciones que tendrá un elemento con otro es a través de las flechas en UML.

Asociación

Esta flecha indica que ese elemento contiene al otro en su definición. La flecha apuntará hacia la dependencia.

Ejemplo:

Se observa como la ClaseA está asociada y depende de la ClaseB.

Herencia

La flecha con la punta sin pintar expresa la herencia. La dirección de la flecha irá desde el hijo hasta el padre.

Ejemplo:

Se observa como la Clase B hereda de la ClaseA.

Agregación

Este es similar a la asociación en que un elemento depende de otro, pero este caso será: Un elemento dependerá de muchos otros.

Se tiene en cuenta la multiplicidad del elemento.

Ejemplo:

Esto indica que la ClaseA contiene varios elementos de la ClaseB. Estos últimos son comúnmente representados con listas o colecciones de datos.

Composición

Su relación es totalmente compenetrada de tal modo que conceptualmente una de estas clases no podría vivir si no existiera la otra.

IV. Objetivos del Proyecto

- **Objetivo General:**

Diseñar 5 objetos diferentes con su correspondiente diagrama UML.

- **Objetivos Específicos:**

- Denotar los atributos y métodos de los cinco objetos.
 - Diseñar el diagrama UML que muestre las relaciones entre las clases.
 - Utilizar conceptos de la programación para estructurar objetos del sistema.
-

V. Metodología

- **Herramientas y Tecnologías Utilizadas:**

Una vez finalizado el proceso de abstracción y se tiene el diseño de clase deseada, el siguiente paso es pasar a un lenguaje de programación. Las transformaciones y las funciones de desarrollo visual permiten a los desarrolladores y a los arquitectos diseñar y desarrollar un modelo de forma concurrente. Los arquitectos crean elementos de modelo conceptuales y, al mismo tiempo, los desarrolladores crean elementos de código o modifican los elementos de código que ha generado una transformación. Las funciones de desarrollo visual permiten a los arquitectos actualizar el modelo UML con elementos de código nuevos. Los arquitectos pueden volver a ejecutar la transformación para generar código para elementos de modelo nuevos.

La estructura general de un programa orientado a objetos depende del lenguaje de programación que se utilice, sin embargo, hay tres principios: clases, objetos y relaciones.

Una aplicación orientada a objetos es una colección de objetos, algunos jerárquicos, que colaboran entre sí para dar una funcionalidad global. Cada objeto debe asumir una parte de las responsabilidades y delegar otras a los objetos colaboradores.

Existirá un objeto que asume las funciones de la aplicación principal, será el encargado de crear las instancias oportunas y tener un único punto de entrada (main)(Bernal, 2012).

- **Procedimiento:**

La clase Estudiante se representa con los atributos necesarios para identificarlo y asociarlo a un curso. Incluye dos atributos principales: ``nombre``, que almacena el nombre completo del estudiante, y ``curso``, un número entero que representa el nivel académico en el que se encuentra.

El constructor inicializa estos atributos al momento de crear el objeto. Además, cuenta con un método llamado ``registrarEstudiante``, que imprime un mensaje en la consola indicando que el estudiante ha sido registrado junto con su nombre y curso. También tiene el método ``consultarInformacion``, que devuelve una descripción completa del estudiante como una cadena de texto.

La clase Notas gestiona las calificaciones del estudiante en distintas materias. Sus datos se almacenan en una estructura ``HashMap``, donde las claves son los nombres de las materias y los valores son las calificaciones correspondientes, representadas como números de tipo ``double``. La clase incluye un constructor que inicializa el `HashMap` vacío y un método ``agregarNota``, que permite añadir calificaciones para una materia específica. Asimismo, el método ``calcularPromedio`` evalúa el promedio de todas las calificaciones registradas y lo devuelve como un valor decimal.

La clase Proyecto modela un proyecto académico asignado a un estudiante. Tiene dos atributos principales: ``titulo``, que contiene el nombre o descripción del proyecto, y ``calificacionProyecto``, que almacena la calificación obtenida por el estudiante. El constructor inicializa ambos atributos, con la calificación establecida inicialmente en cero si no se especifica un valor. El método ``calificarProyecto`` permite asignar una calificación al proyecto, mientras que el método ``consultarProyecto`` devuelve una descripción del proyecto junto con su calificación.

La clase Aprobación evalúa si un estudiante cumple con los criterios necesarios para aprobar. Incluye un atributo booleano llamado ``estado``, que indica si el estudiante está aprobado (``true``) o no (``false``). Su método más importante, ``verificarAprobacion``, recibe como parámetros el promedio de las notas, el porcentaje de asistencia y la calificación del proyecto, y aplica criterios específicos para determinar si el estudiante puede aprobar: el promedio debe ser al menos 60, la asistencia debe ser igual o superior al 75%, y la calificación del proyecto debe ser de al menos 70. El método devuelve ``true`` si se cumplen todos los criterios. Además, el método ``getEstado`` retorna una descripción textual del estado del estudiante, indicando si está "Aprobado" o "No Aprobado".

En conjunto, estas clases interactúan para modelar un sistema académico básico. Cada estudiante está asociado con sus calificaciones, asistencia y proyecto, los cuales son evaluados por la clase de aprobación para determinar si cumple los requisitos académicos. Este diseño permite analizar de forma clara y estructurada el desempeño académico de los estudiantes.

VI. Desarrollo del Proyecto

- **Diseño del Algoritmo:**
 - **Clase Estudiante**

Atributos

nombre: string

curso: int

Métodos

registrarEstudiante(): Registra un nuevo estudiante.

consultarInformacion(): Muestra la información del estudiante.

- **Clase notas**

Atributos

curso: int

calificaciones: lista

promedio: float

Métodos

calcularPromedio(): Calcula el promedio de las notas.

agregarNota(): Añade una nueva calificación

o **Clase asistencia**

Atributos

curso: int

diasAsistencia: int

totalClases: int

Métodos

calcularPorcentaje(): Calcula el porcentaje de asistencia.

registrarAsistencia(): Añade un día asistido.

o **Clase proyecto**

Atributos

curso: int

título: string

calificacionProyecto: Float.

Métodos

calificarProyecto(): Asigna una calificación al proyecto.

consultarProyecto(): Muestra información del proyecto.

o **Clase aprobación**

Atributos

estado: Boolean (aprobado o no).

requisitosCumplidos: Boolean.

Métodos

verificarAprobacion(): Verifica si el estudiante cumple los criterios para aprobar.

Código Fuente:

Para compilar y ejecutar el código tendremos que ejecutar las siguientes sentencias:

```
// Clase Estudiante
public class Estudiante {
    private String nombre;
    private int curso;

    public Estudiante(String nombre, int curso) {
        this.nombre = nombre;
        this.curso = curso;
    }

    public void registrarEstudiante() {
        System.out.println("Estudiante " + nombre + " curso");
    }

    public String consultarInformacion() {
        return "Nombre: " + nombre, "Curso: " + curso;
    }
}

// Clase Notas
import java.util.HashMap;

public class Notas {
    private HashMap<String, Double> calificaciones;

    public Notas() {
        this.calificaciones = new HashMap<>();
    }

    public void agregarNota(String materia, double calificacion) {
        calificaciones.put(materia, calificacion);
    }

    public double calcularPromedio() {
        double total = 0;
        for (double nota : calificaciones.values()) {
            total += nota;
        }
        return calificaciones.size() > 0 ? total / calificaciones.size() : 0;
    }
}
```



```
}
```

// Clase Asistencia

```
public class Asistencia {  
    private int diasAsistidos;  
    private int totalClases;  
  
    public Asistencia(int totalClases) {  
        this.diasAsistidos = 0;  
        this.totalClases = totalClases;  
    }  
  
    public void registrarAsistencia() {  
        diasAsistidos++;  
    }  
  
    public double calcularPorcentaje() {  
        return (diasAsistidos / (double) totalClases) * 100;  
    }  
}
```

// Clase Proyecto

```
public class Proyecto {  
    private String titulo;  
    private double calificacionProyecto;  
  
    public Proyecto(String titulo) {  
        this.titulo = titulo;  
        this.calificacionProyecto = 0;  
    }  
  
    public void calificarProyecto(double calificacion) {  
        this.calificacionProyecto = calificacion;  
    }  
  
    public String consultarProyecto() {  
        return "Proyecto: " + titulo + ", Calificación: " + calificacionProyecto;  
    }  
}
```

// Clase Aprobación

```
public class Aprobacion {  
    private boolean estado;  
  
    public boolean verificarAprobacion(double promedio, double porcentajeAsistencia,  
double calificacionProyecto) {  
        if (promedio >= 60 && porcentajeAsistencia >= 75 && calificacionProyecto >=  
70) {  
            estado = true;  
        } else {  
            estado = false;  
        }  
    }  
}
```

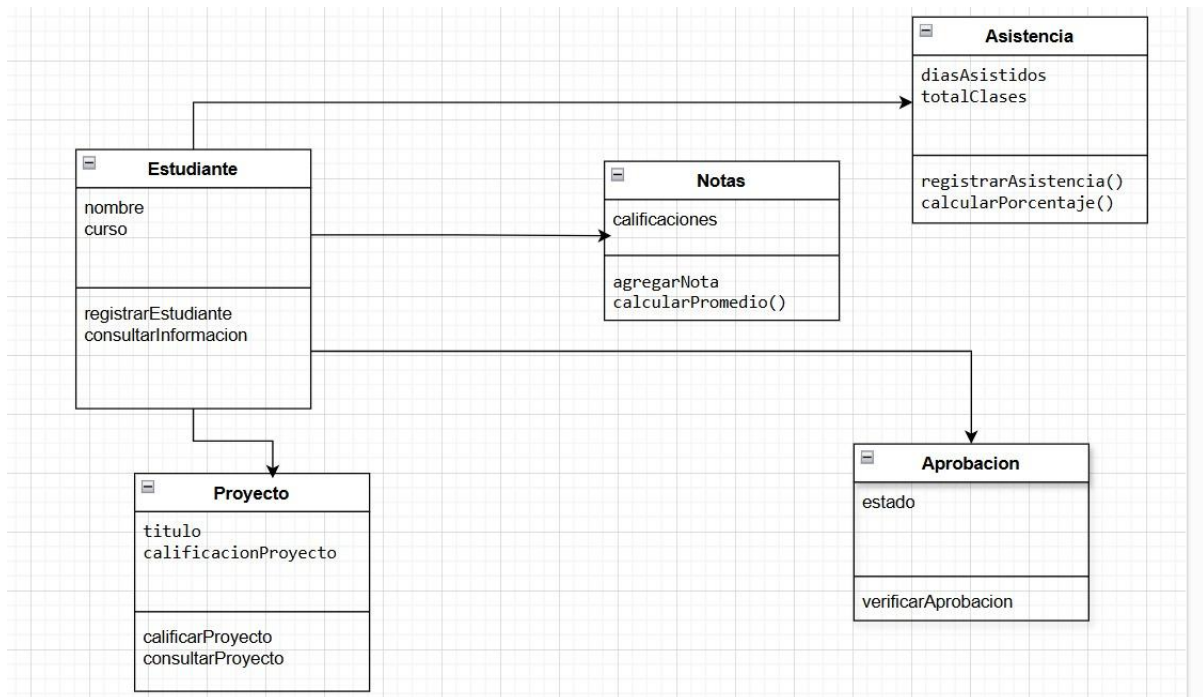
```

    }
    return estado;
}

public String getEstado() {
    return estado ? "Aprobado" : "No Aprobado";
}
}

```

- **Pruebas Realizadas:**



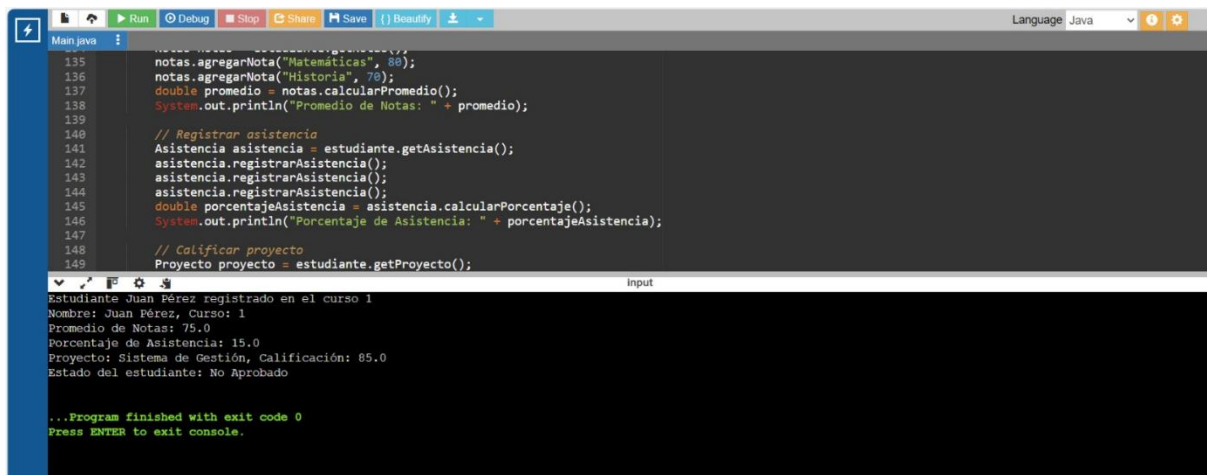
Estudiante: Registra y muestra información del estudiante.

Notas: Permite agregar calificaciones y calcular el promedio.

Asistencia: Registra la asistencia y calcula el porcentaje.

Proyecto: Permite asignar calificación al proyecto.

Aprobación: Verifica si el estudiante cumple con los criterios para aprobar.



```
135 notas.agregarNota("Matemáticas", 88);
136 notas.agregarNota("Historia", 70);
137 double promedio = notas.calcularPromedio();
138 System.out.println("Promedio de Notas: " + promedio);
139
140 // Registrar asistencia
141 Asistencia asistencia = estudiante.getAsistencia();
142 asistencia.registrarAsistencia();
143 asistencia.registrarAsistencia();
144 asistencia.registrarAsistencia();
145 double porcentajeAsistencia = asistencia.calcularPorcentaje();
146 System.out.println("Porcentaje de Asistencia: " + porcentajeAsistencia);
147
148 // Calificar proyecto
149 Proyecto proyecto = estudiante.getProyecto();
```

Estudiante Juan Pérez registrado en el curso 1
Nombre: Juan Pérez, curso: 1
Promedio de Notas: 75.0
Porcentaje de Asistencia: 15.0
Proyecto: Sistema de Gestión, Calificación: 85.0
Estado del estudiante: No Aprobado

...Program finished with exit code 0
Press ENTER to exit console.

VII. Resultados

- **Análisis de Resultados:**
 - Se diseñó un sistema funcional orientado a objetos que representa la gestión académica básica de estudiantes, utilizando clases como Estudiante, Notas, Proyecto, y Aprobación.
 - El sistema permite registrar datos de estudiantes, gestionar sus calificaciones, proyectos, y asistencia, y determinar automáticamente si un estudiante cumple con los criterios para aprobar.
 - Se utilizó el paradigma de programación orientada a objetos para implementar principios clave como encapsulación, modularidad y reutilización de código.

VIII. Conclusiones

- **Conclusiones Generales:**
 - El proyecto desarrollado demuestra que la programación orientada a objetos es una herramienta eficaz para modelar y resolver problemas del mundo real, como la gestión académica de estudiantes. A través del diseño y la implementación de clases como Estudiante, Notas, Proyecto y Aprobación, se logró estructurar un sistema modular que facilita el manejo de información y

la automatización de procesos. El uso de relaciones entre clases, como la asociación y la dependencia, permitió construir un modelo flexible y escalable, que puede adaptarse a futuros requerimientos.

Además, se pudo constatar la utilidad de encapsular atributos y métodos en cada clase, lo que permitió un manejo eficiente y seguro de los datos. El sistema diseñado no solo permitió registrar información relevante de los estudiantes, como sus calificaciones y proyectos, sino también evaluar automáticamente su desempeño académico con base en criterios predefinidos. Esto resalta la capacidad del proyecto para automatizar decisiones, ahorrando tiempo y reduciendo errores humanos en procesos de evaluación.

- **Recomendaciones:**

- Ampliar los atributos de la clase para incluir más información del estudiante, como correo electrónico, teléfono, y otros datos de contacto.
- Incorporar una base de datos para almacenar de forma permanente la información de los estudiantes, sus calificaciones, asistencia y proyectos.

IX. Referencias

Bermón, L. (2021). Ejercicios de programación orientada a objetos con Java y UML.

Universidad Nacional de Colombia.

https://fadmon.unal.edu.co/fileadmin/user_upload/investigacion/centro_editorial/libros/ejercicios%20de%20programacion.pdf

Codigofacilito. (2010). UML y diagrama de clases.

https://codigofacilito.com/articulos/uml_diagramas_de_clase

Jaramillo, L. (11 agosto 2024). Programación Orientada a Objetos.

<https://luisjaramillom.github.io/POO.io/>

Vélez S., Jose; Peña, A; Gortazar, F; Sánchez, A. (2011). Diseñar y programar, todo es empezar: una introducción a la Programación Orientada a Objetos usando UML y Java. Madrid. Dykinson.

<http://www.visionporcomputador.es/libroSoftware/Tecnicas%20avanzadas%20de%20diseno%20de%20software.pdf>