Servicio Nacional de Aprendizaje – SENA

Centro de Diseño y Metrología

Aprendiz:

Edwin Camilo Lozano Chaparro

No Ficha 2848530-A

Introducción

En este informe se presentan una serie de implementaciones en Java diseñadas para abordar la gestión de empleados y estudiantes, demostrando un enfoque tanto estructurado como orientado a objetos. A través de los ejemplos, se destacan conceptos esenciales de programación como el uso de colecciones, la encapsulación, la reutilización de código mediante clases y métodos, y la interacción con el usuario a través de la consola.

En el primer conjunto de programas, enfocados en la gestión de empleados, se muestran dos enfoques: uno estructurado, utilizando arreglos dinámicos para almacenar información, y otro basado en la Programación Orientada a Objetos (POO), donde las entidades como Empleado y Empresa encapsulan datos y lógica relacionada. Por otro lado, los programas relacionados con estudiantes también exploran ambos paradigmas para calcular promedios de notas, determinando si los estudiantes aprueban o reprueban.

Estos códigos no solo demuestran el uso de estructuras básicas como bucles y condicionales, sino que también introducen técnicas avanzadas como la interacción con múltiples clases, encapsulación de datos y modularidad, contribuyendo a sistemas más claros, robustos y escalables.

Contenido

códigos de programación estructurada

códigos empleados estructurada

```
import java.util.ArrayList;
import java.util.Scanner;


public class EmpleadosEstructurada{

  public static void main(String[] args) {

    String nombre, cargo;

    double salario, total = 0, minSalario, maxSalario;

    int cantidad, posicion = 0;


    ArrayList<String> nombres = new ArrayList<String>();

    ArrayList<Double> salarios = new ArrayList<Double>();

    ArrayList<String> cargos = new ArrayList<String>();

    Scanner cantidades = new Scanner(System.in);
    Scanner names = new Scanner(System.in);
    Scanner salari= new Scanner(System.in);
    Scanner carg = new Scanner(System.in);

     System.out.println("Ingresar cantidad de empleados");
     cantidad = cantidades.nextInt();

    for (int i =1 ; i <= cantidad ; i++) {

          System.out.println("Ingrese el nombre del empleado");
          nombre = names.next();
          System.out.println("Ingrese el cargo del empleado");
          cargo = carg.next();
          System.out.println("Ingrese el salario del empleado");
          salario = salari.nextDouble();


          nombres.add(nombre);
          salarios.add(salario);
```

```
          cargos.add(cargo);

     }
    System.out.println("El total de empleados es: " + cantidad);

    System.out.println("Los nombres y salarios son: ");


    for (int i =0; i < cantidad ; i++) {
        System.out.println("Nombres: " + nombres.get(i) + " Salarios: " +
salarios.get(i));

      }

    for (int i =0; i < cantidad ; i++) {

          total = total + salarios.get(i);

      }

      System.out.println("El total de salarios pagados es de: " +total);

      System.out.println("El empleado que mas dinero gana es: ");
      maxSalario = salarios.get(0);

      for (int i =1; i < cantidad ; i++) {
          if(salarios.get(i) > maxSalario) {
              maxSalario = salarios.get(i);
              posicion = i;
          }
      }

      System.out.println("Nombre: " + nombres.get(posicion) + " Cargo: " +
cargos.get(posicion) + " Salario: " + salarios.get(posicion));

      System.out.println("El empleado que menos dinero gana es: ");
      minSalario = salarios.get(0);


      for (int i =1; i < cantidad ; i++) {

          if(salarios.get(i) < minSalario) {
              minSalario = salarios.get(i);
              posicion = i;
          }
```

```
      }

      System.out.println("Nombre: " + nombres.get(posicion) + " Cargo: " +
cargos.get(posicion) + " Salario: " + salarios.get(posicion));

  }

}
```

Explicación del código:

Este programa en Java se utiliza para gestionar y analizar los datos de un grupo de empleados, incluyendo sus nombres, cargos y salarios.

Introducción al código:

- **Bibliotecas utilizadas:**
    - ArrayList: Para manejar colecciones dinámicas de datos como nombres, cargos y salarios.
    - Scanner: Para capturar la entrada del usuario desde la consola.

Estructura y funcionamiento del código:

1. **Declaración de variables:**
    - nombre, cargo: Strings para almacenar los datos del empleado.
    - salario, total, minSalario, maxSalario: Variables de tipo double para manejar información financiera.
    - cantidad, posicion: Variables int para manejar la cantidad de empleados y la posición del empleado con mayor o menor salario.
2. **Inicialización de listas:**
    - Se utilizan ArrayList para almacenar:
        - nombres de los empleados.
        - cargos de los empleados.
        - salarios de los empleados.
3. **Captura de datos:**
    - Solicita al usuario la cantidad de empleados.
    - Recolecta datos de cada empleado a través de bucles e interfaces de entrada (Scanner).
4. **Procesamiento de datos:**

- Se calcula el total de salarios sumando los valores almacenados en la lista de salarios.
- Identifica al empleado con el salario más alto y al de menor salario mediante comparaciones iterativas.

5. **Salida de resultados:**
   - Muestra en consola:
     - El total de empleados registrados.
     - Nombres y salarios de los empleados.
     - Total de salarios pagados.
     - Detalles del empleado con el mayor salario.
     - Detalles del empleado con el menor salario.

Puntos clave:

- Uso de estructuras de datos dinámicas (ArrayList) para organizar y gestionar los datos de manera flexible.
- Eficiencia en el cálculo de estadísticas como el total, el salario máximo y mínimo mediante iteraciones.

Código notas estructurada

```
import java.util.Scanner;

public class NotasEstructurada {
    public static void main(String[] args) throws Exception {

        String name;
        Double noteParcial1;
        Double noteParcial2;
        Double noteFinal;

        Scanner nombre = new Scanner(System.in);
        Scanner nota1 = new Scanner(System.in);
        Scanner nota2 = new Scanner(System.in);

        for (int i = 1; i < 4; i++) {
            System.out.println("Ingrese el nombre del " + i + " Estudiante:
");
```

```
            name = nombre.next();
            System.out.println("Ingrese la nota 1 del parcial del " + i + "
Estudiante: ");
            noteParcial1 = nota1.nextDouble();
            System.out.println("Ingrese la nota 2 del parcial del " + i + "
Estudiante: ");
            noteParcial2 = nota2.nextDouble();

            noteFinal = (noteParcial1 + noteParcial2) / 2;

            System.out.println("Informacion del Estudiante: " + name + "
Nota 1: " + noteParcial1 + " Nota 2: "
                    + noteParcial2 + "  Resultado del promedio de las 2
notas: " + noteFinal);

            if (noteFinal < 3) {

                System.out.println("Usted: " + name + " Reprobo");

            } else {
                System.out.println("Usted: " + name + " Aprobo");
            }

        }

    }
}
```

Explicación del código:

El programa **NotasEstructurada** es una aplicación simple en Java que permite ingresar y procesar las calificaciones de tres estudiantes, calculando su promedio final y determinando si aprobaron o reprobaron.

Introducción al código:

- **Objetivo principal:** Procesar los datos académicos de estudiantes.
- **Bibliotecas utilizadas:**
    - Scanner: Para capturar las entradas del usuario desde la consola.

Estructura y funcionamiento del código:

1. **Declaración de variables:**
    - name: Para almacenar el nombre de cada estudiante.

- noteParcial1, noteParcial2: Notas parciales del estudiante.
- noteFinal: Almacena el promedio de las dos notas.

2. **Inicialización de Scanner:**
   - Tres objetos Scanner se utilizan para manejar diferentes entradas de datos:
     - nombre: Captura el nombre del estudiante.
     - nota1, nota2: Capturan las notas parciales.
3. **Bucle para capturar datos y calcular resultados:**
   - Se utiliza un for que itera tres veces para procesar información de tres estudiantes.
   - En cada iteración:
     - Solicita el nombre y las dos notas parciales.
     - Calcula el promedio (noteFinal = (noteParcial1 + noteParcial2) / 2).
     - Muestra la información del estudiante: nombre, notas parciales y promedio.
     - Evalúa si el estudiante aprobó o reprobó con base en el promedio:
       - Si el promedio es menor que 3, se considera reprobado.
       - En caso contrario, se considera aprobado.
4. **Salida de resultados:**
   - Se imprime un resumen con los datos de cada estudiante y el resultado de su evaluación (aprobado o reprobado).

Puntos clave:

- Uso de un bucle for para procesar datos repetitivos, en este caso, de tres estudiantes.
- Condicionales if-else para evaluar y clasificar los resultados según el promedio.
- Uso de múltiples objetos Scanner para diferenciar entradas de datos por tipo o contexto.

Códigos programación orientada a objetos

Código empleado POO

```java
import java.util.Scanner;

public class ejercicioEmpleadoPOO2 {
    public static void main(String[] args) throws Exception {
```

```
        int cantidad;
        String nombre;
        String cargo;
        double salario;

        Empresa empresa = new Empresa();

        Scanner cantidades = new Scanner(System.in);

        Scanner nombres = new Scanner(System.in);

        Scanner cargos = new Scanner(System.in);

        Scanner salarios = new Scanner(System.in);

        System.out.println("Ingrese cantidad de empleados");
        cantidad = cantidades.nextInt();

        for (int i = 0; i <= cantidad; i++) {
            System.out.println("Ingresar nombre del empleado");
            nombre = nombres.next();
            System.out.println("Ingresar cargo del empleado");
            cargo = cargos.next();
            System.out.println("Ingresar salario del empleado");
            salario = salarios.nextInt();
            empresa.contratarEmpleado(new Empleado(cargo, nombre, salario));

        }

        System.out.println("Numero total de empleados: " +
empresa.getTotalEmpleados());

        System.out.println("Nombres y salarios de los empleados son: ");
        empresa.nombreSalario();

        System.out.println("Total de dinero pagado a todos es de: " +
empresa.getTotalSalarios());

        empresa.empleadoMayorSalario();
        empresa.empleadoMenorSalario();
    }

}
```

Explicación del código:

El programa **ejercicioEmpleadoPOO2** es una solución orientada a objetos para gestionar información de empleados en una empresa. Permite registrar empleados, calcular el total de salarios, identificar quién gana más o menos, y mostrar datos de manera estructurada.

Introducción al código:

- **Objetivo principal:** Implementar un sistema de gestión de empleados que registre información básica y realice cálculos relacionados.
- **Enfoque:** Uso de clases y objetos para estructurar la lógica y datos.

Estructura y funcionamiento del código:

1. **Declaración de variables:**
    - cantidad: Cantidad de empleados a registrar.
    - nombre, cargo, salario: Datos básicos de cada empleado.
2. **Clases utilizadas:**
    - **Empresa:** Representa la empresa que gestiona los empleados.
    - **Empleado:** Representa a cada empleado con sus atributos como cargo, nombre y salario.
3. **Inicialización de objetos Scanner:**
    - Captura entradas del usuario para la cantidad de empleados y los datos de cada empleado (nombre, cargo, salario).
4. **Registro de empleados:**
    - Se solicita al usuario la cantidad de empleados a registrar.
    - En un bucle for, se captura la información de cada empleado.
    - Cada empleado se registra en la empresa mediante el método contratarEmpleado.
5. **Cálculos y reportes:**
    - **Total de empleados:** Se muestra el número total de empleados registrados.
    - **Nombres y salarios:** Llama al método nombreSalario de la clase Empresa para listar los datos de cada empleado.
    - **Total de salarios:** Calcula y muestra la suma total de salarios pagados por la empresa.
    - **Mayor y menor salario:** Identifica y muestra qué empleados ganan el mayor y el menor salario respectivamente, utilizando métodos en la clase Empresa.

Puntos clave del diseño:

- Uso de programación orientada a objetos para organizar los datos y la lógica del programa.
- Separación de responsabilidades:
    - La clase Empleado encapsula los datos básicos de un empleado.
    - La clase Empresa maneja la lista de empleados y realiza operaciones sobre ellos, como calcular salarios totales y encontrar el salario más alto o más bajo.
- Uso de estructuras de control como bucles para registrar empleados dinámicamente según la cantidad especificada por el usuario

Código clase empleado POO

```
public class Empleado {
    public String nombre;
    public String cargo;
    public double salario;

    public Empleado(String cargo, String nombre, double salario) {
        this.cargo = cargo;
        this.nombre = nombre;
        this.salario = salario;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getCargo() {
        return cargo;
    }

    public void setCargo(String cargo) {
        this.cargo = cargo;
    }

    public double getSalario() {
        return salario;
    }

    public void setSalario(double salario) {
```

```
        this.salario = salario;
    }
}
```

Explicación del código:

La clase **Empleado** representa un modelo de datos para almacenar información básica de un empleado, como su nombre, cargo y salario. Es una parte esencial de la solución orientada a objetos para gestionar empleados dentro de una empresa.

Detalles del código:

1. **Atributos:**
    - nombre (String): Nombre del empleado.
    - cargo (String): Cargo que desempeña el empleado.
    - salario (double): Salario asignado al empleado.
2. **Constructor:**
    - Empleado(String cargo, String nombre, double salario): Inicializa un objeto de tipo empleado con los valores proporcionados para cargo, nombre y salario.
3. **Métodos:**
    - **Getters:** Métodos que permiten acceder a los valores de los atributos:
        - getNombre(): Retorna el nombre del empleado.
        - getCargo(): Retorna el cargo del empleado.
        - getSalario(): Retorna el salario del empleado.
    - **Setters:** Métodos que permiten modificar los valores de los atributos:
        - setNombre(String nombre): Asigna un nuevo valor al nombre del empleado.
        - setCargo(String cargo): Asigna un nuevo valor al cargo del empleado.
        - setSalario(double salario): Asigna un nuevo valor al salario del empleado.

Importancia de esta clase:

- **Encapsulación:** Permite encapsular los datos relacionados con un empleado en una sola entidad.
- **Reutilización:** Es reutilizable en diferentes contextos dentro del sistema, como la gestión de empleados en una empresa.

- **Flexibilidad:** Con el uso de getters y setters, los atributos del empleado pueden ser leídos y modificados de manera controlada.

Esta clase será utilizada en combinación con otras clases, como la clase Empresa, para realizar operaciones sobre una lista de empleados, como calcular salarios totales o identificar al empleado con mayor o menor salario.

Código empresa POO

```
import java.util.ArrayList;

public class Empresa {

    public ArrayList<Empleado> empleados;

    public Empresa() {
        empleados = new ArrayList<Empleado>();
    }

    public void contratarEmpleado(Empleado e) {
        empleados.add(e);
    }

    public int getTotalEmpleados() {
        return empleados.size();
    }

    public void nombreSalario() {
        for (Empleado e : empleados) {
            System.out.println("Nombre: " + e.getNombre() + " Salario: " +
e.getSalario());
        }
    }

    public double getTotalSalarios() {
        double total = 0;
        for (Empleado e : empleados) {
            total = total + e.getSalario();
        }
        return total;
    }

    public void empleadoMayorSalario() {
        Empleado empMayorSalario = empleados.get(0);
        double maxSalario = empleados.get(0).getSalario();
```

```
        for (Empleado e : empleados) {
            if (e.getSalario() > maxSalario) {
                maxSalario = e.getSalario();
                empMayorSalario = e;
            }
        }
        System.out.println("El empleado que mas dinero gana es: ");
        System.out.print("Nombre: " + empMayorSalario.getNombre());
        System.out.print(" Cargo: " + empMayorSalario.getCargo());
        System.out.println(" Salario: " + empMayorSalario.getSalario());
    }

    public void empleadoMenorSalario() {
        Empleado empMenorSalario = empleados.get(0);
        double menSalario = empleados.get(0).getSalario();

        for (Empleado e : empleados) {
            if (e.getSalario() < menSalario) {
                menSalario = e.getSalario();
                empMenorSalario = e;
            }
        }
        System.out.println("El empleado que menos dinero gana es: ");
        System.out.print("Nombre: " + empMenorSalario.getNombre());
        System.out.print(" Cargo: " + empMenorSalario.getCargo());
        System.out.println(" Salario: " + empMenorSalario.getSalario());
    }

}
```

Explicación del código:

La clase **Empresa** modela una entidad para gestionar un grupo de empleados y sus atributos. Utiliza una lista para almacenar los empleados y proporciona métodos para realizar diversas operaciones relacionadas con ellos, como la contratación, el cálculo de salarios y la búsqueda del empleado con mayor o menor salario.

Detalles del código:

1. **Atributos:**
    - empleados (ArrayList<Empleado>): Lista que almacena objetos de tipo **Empleado**, permitiendo la gestión de múltiples empleados en la empresa.
2. **Constructor:**
    - Empresa(): Inicializa la lista empleados como una nueva instancia de ArrayList.

3. **Métodos principales:**
    - **Contratar empleado:**
        - contratarEmpleado(Empleado e): Añade un empleado (objeto de tipo Empleado) a la lista de empleados de la empresa.
    - **Total de empleados:**
        - getTotalEmpleados(): Retorna el número total de empleados contratados en la empresa.
    - **Nombres y salarios:**
        - nombreSalario(): Imprime el nombre y el salario de cada empleado en la lista.
    - **Total de salarios:**
        - getTotalSalarios(): Calcula y retorna la suma de los salarios de todos los empleados.
    - **Empleado con mayor salario:**
        - empleadoMayorSalario(): Identifica al empleado con el salario más alto e imprime su información (nombre, cargo y salario).
    - **Empleado con menor salario:**
        - empleadoMenorSalario(): Identifica al empleado con el salario más bajo e imprime su información (nombre, cargo y salario).

Características clave:

- **Uso de listas dinámicas:** El uso de ArrayList permite agregar empleados de forma dinámica, sin límite predefinido.
- **Operaciones iterativas:** Métodos como empleadoMayorSalario() y empleadoMenorSalario() recorren la lista de empleados para realizar comparaciones y extraer resultados específicos.
- **Modularidad:** Cada funcionalidad está encapsulada en un método específico, facilitando la legibilidad y mantenimiento del código.
- **Relación con Empleado:** Esta clase depende de la clase Empleado, lo que demuestra un diseño orientado a objetos basado en relaciones entre clases.

Importancia de la clase:

La clase **Empresa** actúa como un controlador para las operaciones relacionadas con los empleados. Representa una solución estructurada para gestionar datos de empleados y realizar cálculos como el total de salarios o la búsqueda de empleados destacados por sus remuneraciones

Código estudiante POO

```
. import java.util.Scanner;

public class ejercicioEstudiantePOO2 {
    public static void main(String[] args) throws Exception {

        Estudiante est;

        String nombre;
        double nota1, nota2;
        Scanner name = new Scanner(System.in);
        Scanner not1 = new Scanner(System.in);
        Scanner not2 = new Scanner(System.in);

        for (int i = 1; i < 4; i++) {
            System.out.println("Ingrese el nombre del " + i + " Estudiante:
");
            nombre = name.next();
            System.out.println("Ingrese la nota 1 del parcial del " + i + "
Estudiante: ");
            nota1 = not1.nextDouble();
            System.out.println("Ingrese la nota 2 del parcial del " + i + "
Estudiante: ");
            nota2 = not2.nextDouble();

            est = new Estudiante(nombre);
            est.asignarNota1(nota1);
            est.asignarNota2(nota2);
            est.calcularNotaFinal();

            System.out.println("Querido estudiante: " + est.nombre + " Su
nota 1 fue de: " + est.obtenerNota1()
                    + " Su nota 2 fue de: " + est.obtenerNota2() + " El
promedio de sus 2 notas parciales es de: "
                    + est.obtenerNotaFinal() + " Usted" +
est.obtenerMensaje());

        }
    }
}
```

Explicación del código:

El programa **ejercicioEstudiantePOO2** implementa una solución en programación orientada a objetos para procesar información de estudiantes y calcular sus promedios con base en dos notas parciales. Utiliza la clase **Estudiante** como base para manejar los datos y operaciones relacionadas con cada estudiante.

Detalles del código:

1. **Objetivo principal:**
    - Procesar la información de 3 estudiantes, incluyendo sus nombres y dos notas parciales, para calcular y mostrar el promedio final junto con un mensaje de aprobación o reprobación.
2. **Variables locales:**
    - nombre (String): Almacena el nombre del estudiante.
    - nota1, nota2 (double): Almacenan las notas parciales de cada estudiante.
    - est (Estudiante): Representa un objeto de tipo **Estudiante** que se instancia en cada iteración del bucle.
3. **Entrada de datos:**
    - Se utilizan instancias de Scanner para capturar la información ingresada por el usuario, como el nombre del estudiante y sus dos notas parciales.
4. **Flujo del programa:**
    - El programa utiliza un bucle for para procesar los datos de 3 estudiantes.
    - En cada iteración:
        - Se solicita el nombre y las dos notas del estudiante.
        - Se crea un objeto **Estudiante** con el nombre proporcionado.
        - Las notas se asignan mediante los métodos de la clase **Estudiante**.
        - Se calcula la nota final usando el método calcularNotaFinal().
5. **Salida de datos:**
    - Para cada estudiante, se muestra un mensaje que incluye:
        - Nombre del estudiante.
        - Notas parciales.
        - Promedio de las notas.
        - Mensaje indicando si aprobó o reprobó, según su promedio.

Interacción con la clase Estudiante:

La clase **Estudiante** (no incluida en el código proporcionado) se encarga de encapsular las propiedades y métodos asociados con el estudiante, incluyendo:

- Asignar y obtener notas parciales.
- Calcular el promedio.
- Generar un mensaje basado en el promedio (aprobación/reprobación).

Características destacadas:

1. **Orientación a objetos:**
    - El uso de la clase **Estudiante** mejora la organización del código y su reutilización.
2. **Estructura modular:**
    - Las operaciones de entrada, cálculo y salida están separadas, lo que facilita la comprensión del código.
3. **Mensajes personalizados:**
    - Se generan mensajes específicos para cada estudiante, lo que mejora la interacción con el usuario.

Importancia del programa:

El programa **ejercicioEstudiantePOO2** demuestra un enfoque claro para manejar datos estudiantiles en un entorno estructurado y orientado a objetos. Es ideal para aplicaciones educativas o administrativas donde se necesita gestionar información de estudiantes y evaluar su desempeño.

Código clase estudiante POO

```
public class Estudiante {

	public String nombre;
	public double nota1;
	public double nota2;
	public double notaFinal;

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

	public Estudiante(String nombre, double np1, double np2) {
		this.nombre = nombre;
		nota1 = np1;
```

```
                nota2 = np2;
        }

        public void asignarNota1(double np1) {
                nota1 = np1;
        }

        public void asignarNota2(double np2) {
                nota2 = np2;
        }

        public double obtenerNota1() {
                return nota1;
        }

        public double obtenerNota2() {
                return nota2;
        }

        public void calcularNotaFinal() {
                notaFinal = (nota1 + nota2) / 2;
        }

        public double obtenerNotaFinal() {

                return notaFinal;
        }

        public String obtenerMensaje() {
                if (notaFinal < 3)
                        return " ha reprobado lo lamento";
                else
                        return " ha aprobado felicidades";
        }
}
```

La clase **Estudiante** está diseñada para gestionar información básica sobre un estudiante y calcular su desempeño académico en base a dos notas parciales. Proporciona métodos para asignar valores, realizar cálculos y generar mensajes personalizados según los resultados obtenidos.

Detalles del código:

1. **Atributos:**
    - nombre (String): Nombre del estudiante.

- nota1, nota2 (double): Notas parciales del estudiante.
- notaFinal (double): Promedio de las dos notas parciales.

2. **Constructores:**
   - **Constructor con nombre:** Inicializa el nombre del estudiante.
   - **Constructor completo:** Inicializa el nombre y las dos notas parciales, útil para crear instancias con todos los datos desde el inicio.
3. **Métodos principales:**
   - **Asignación de notas:**
     - asignarNota1(double np1) y asignarNota2(double np2) asignan valores a las notas parciales.
   - **Obtención de datos:**
     - obtenerNota1(), obtenerNota2(): Devuelven las notas parciales.
     - obtenerNotaFinal(): Devuelve el promedio calculado.
   - **Cálculo de la nota final:**
     - calcularNotaFinal(): Calcula y asigna el promedio de las dos notas parciales al atributo notaFinal.
   - **Generación de mensaje personalizado:**
     - obtenerMensaje(): Evalúa si el estudiante aprobó o reprobó en base a la nota final (aprobado si el promedio es mayor o igual a 3).
4. **Mensajes dinámicos:**
   - El método obtenerMensaje() genera un mensaje basado en el promedio:
     - Si el promedio es menor a 3, el estudiante ha reprobado.
     - Si el promedio es mayor o igual a 3, el estudiante ha aprobado.

Características destacadas:

1. **Encapsulación parcial:**
   - Aunque los atributos son públicos, los métodos proporcionan control sobre cómo se accede y manipulan los datos.
2. **Constructores flexibles:**
   - Permiten inicializar objetos de diferentes maneras según los datos disponibles.
3. **Reutilización:**

- Métodos como calcularNotaFinal() y obtenerMensaje() encapsulan lógica específica, lo que facilita su reutilización en otros contextos.

Importancia en el programa principal:

La clase **Estudiante** se utiliza en el programa **ejercicioEstudiantePOO2** para gestionar las operaciones relacionadas con estudiantes:

- Almacena los datos individuales de cada estudiante.
- Calcula los promedios de manera precisa.
- Genera mensajes personalizados para interactuar con el usuario.

Gracias a esta clase, el programa principal puede enfocarse en la lógica de flujo y entrada/salida, delegando las operaciones específicas a los métodos de la clase. Esto mejora la modularidad, la claridad y la escalabilidad del código.

Conclusión

A través del desarrollo de los códigos presentados, se evidencian las ventajas y diferencias entre los enfoques estructurado y orientado a objetos. Mientras que el enfoque estructurado es útil para soluciones rápidas y directas, el uso de POO permite una mayor organización, escalabilidad y reutilización del código, haciendo que el mantenimiento y la expansión del software sean más sencillos.

En la gestión de empleados, la implementación orientada a objetos destaca por su capacidad para manejar múltiples empleados de manera estructurada, facilitando operaciones como el cálculo de salarios totales y la identificación de empleados con salarios extremos. De manera similar, en el caso de los estudiantes, la POO permite encapsular la lógica de cálculo de notas y generar mensajes personalizados, dejando al programa principal la responsabilidad de gestionar el flujo de la aplicación.

En conjunto, estos programas destacan por su claridad, eficiencia y capacidad de adaptación, reflejando buenas prácticas de programación y una base sólida para abordar problemas más complejos en el futuro.