República Bolivariana de Venezuela
Ministerio del Poder Popular para la Educación
Universitaria Universidad Politécnica Territorial de Carcas
"Mariscal de Sucre"
PNF Informática
Sección 701201 – Trayecto I – Trimestre III
Algoritmo y Programación

Informe - Proyecto 3

Sistema Avanzado de Gestión de Personal y Proyectos

Alumna:
Naomy Vaamondes V-27793827
Profesor:
José Gregorio Ortiz

Índice

3

Introducción

El proyecto a continuación es un Sistema de Gestión de Personal y Proyectos, se realizó en lenguaje Python y siguiendo exclusivamente las indicaciones del profesor y con el conocimiento anteriormente adquirido en clases sobre la Programación Orientada a objetos.

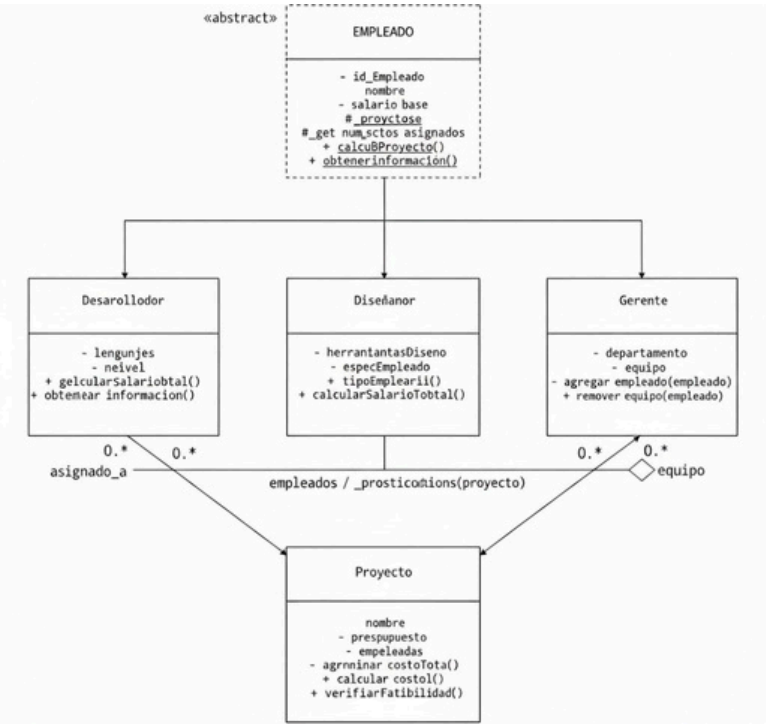
La finalidad de dicho proyecto fue cumplir con el objetivo principal que nos indicó el enunciado: ***"crear para una empresa de tecnología un sistema orientado a objetos para gestionar su personal y los proyectos en los que participan"***. Para esto utilicé, herencia, composición, encapsulamiento y polimorfismo.

Realicé un caso base de prueba para asegurarme de que el sistema fuese capaz de:

- Crear diferentes tipos de empleados (Desarrollador, Diseñador, Gerente).
- Calcular el salario total de cada empleado según reglas específicas (Polimorfismo).
- Asignar empleados a proyectos, respetando un límite de participación por tipo de rol.
- Evaluar la viabilidad financiera de los proyectos.

4

Diagrama de clases



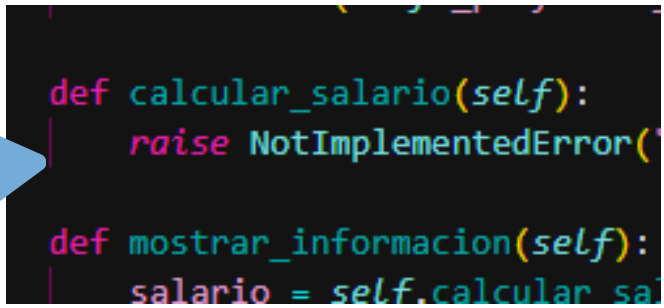
Se me dificultó realizar el diagrama así que pedí ayuda a la IA.

5

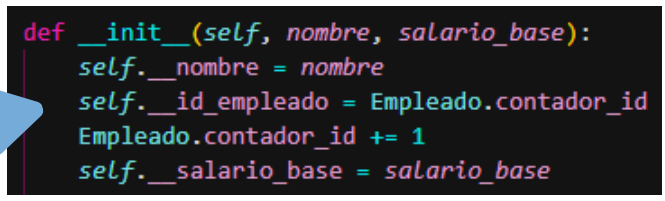
Implementación de clases

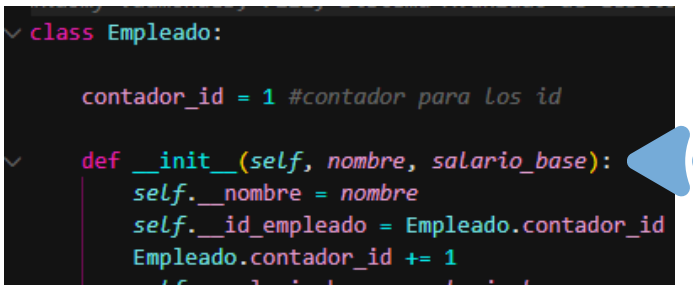
Clase base: Empleados

Esta clase es la principal, y la que me sirvió como guía para crear a todas las demás. En esta clase implementé la abstracción en el metodo calcular_salario, con un raise (lanza error), para que así cuando las clases hijas tengan que calcular su salario funcione correctamente.



También se aplicó el encapsulamiento mediante los __, esto para que así los atributos fuesen privados como se pidió en el enunciado.





Para crear los ID realicé la variable contador_id, esta existe 1 sola vez en la clase empleados, esto ya que las otras clases son heredadas de esta clase y al llamarla se suma un 1 al contador de ID, así es seguro que cada empleado tendrá un ID único.

6

Implementación de clases

Clases: Desarrollador, Diseñador y Gerente

En estas clases se puede ver que se aplica la herencia, ya que heredan de la clase empleado. Y a su vez aplican el polimorfismo cuándo calculan su propio salario, ya que hacen la misma acción pero generan resultados diferentes.

En las clases Desarrollador y Diseñador para calcular los salarios de aplicaron los bonos fijos que indicó en el enunciado.

En la clase Gerente debido al bono de 15% del total de salarios de su equipo, fue necesario llamar al método calcular_salario de cada empleado de la lista equipo.

Herencia

```
class Desarrollador(Empleado):

    def __init__(self,
        super().__init__
        self.lenguajes = leng
        self.nivel = nivel
```

```
def calcular_salario(self):
    salario = self.get_salario_base()
    bono = 0.0
    if self.nivel == "Junior":
        bono = 200.0
    elif self.nivel == "SemiSenior":
        bono = 500.0
    elif self.nivel == "Senior":
        bono = 1000.0
    return salario + bono
```

Clase Desarrollador

```
def calcular_salario(self):
    salario = self.get_salario_base()
    total_salarios_equipo = sum(emp.calcular_salario() for emp in self.equipo)
    bono = total_salarios_equipo * 0.15
    return salario + bono
```

Clase Gerente

Implementación de clases

Clase: Proyecto

En esta clase se aplicó la composición al tener como atributo a la lista de empleados asignados, ya que se está vinculando el proyecto con los empleados asignados.

El método viabilidad cumple la regla de negocio, retornando True cuando el costo_total no supera el 70% del presupuesto.

Clase Proyecto

```python
class Proyecto:

    def __init__(self, nombre, presupuesto):
        self.__nombre = nombre
        self.__presupuesto = presupuesto
        self.empleados = []
```

```python
def viabilidad(self):
    costo = self.costo_total()
    viable = costo <= (self.__presupuesto * 0.7)
    print("Proyecto " + self.__nombre)
    print("Costo total " + str(costo))
    print("Presupuesto maximo viable " + str(self.__presupuesto * 0.7))
    print("Viabilidad " + str(viable))
    return viable
```

Clase Proyecto

8

Caso prueba

Para probar el sistema se realizó el caso prueba:

- ◯ 1 gerente con 2 desarrolladores y 1 diseñador en su equipo.
- ◯ 2 proyectos con presupuestos definidos.
- ◯ Asigna empleados a los proyectos respetando sus límites.
- ◯ Muestra la viabilidad de cada proyecto.
- ◯ Intenta asignar un cuarto proyecto a un desarrollador y captura el error.

Lo añadí al código para ver cómo se comportaba

```
#caso de purruebaaa!
print("-Inicializacion del Sistema")
gerente1 = Gerente("Aisha Herrera", 6000.0, "Desarrollo de Software")
desarrollador1 = Desarrollador("Carlos Guevara", 3000.0, ["Python", "SQL"], "SemiSenior")
desarrollador2 = Desarrollador("Alber Curvelo", 2500.0, ["Java"], "Junior")
disenador1 = Diseñador("Hanna Montana", 3200.0, ["Figma", "Photoshop", "Illustrator"], "UX")
empleados_general = [gerente1, desarrollador1, desarrollador2, disenador1]
gerente1.agregar_al_equipo(desarrollador1)
gerente1.agregar_al_equipo(desarrollador2)
gerente1.agregar_al_equipo(disenador1)
gerente1.agregar_al_equipo(gerente1)
print("-Salarios calculados (antes de proyectos)")
procesar_empleados(empleados_general)
proyectoA = Proyecto("AppMovil", 30000.0)
proyectoB = Proyecto("SistemaWeb", 15000.0)
print("-Asignacion de Proyectos")
proyectoA.agregar_empleado(desarrollador1)
proyectoB.agregar_empleado(desarrollador1)
proyectoA.agregar_empleado(desarrollador2)
proyectoA.agregar_empleado(disenador1)
proyectoB.agregar_empleado(disenador1)
proyectoA.agregar_empleado(gerente1)
print("-Viabilidad de Proyectos")
proyectoA.viabilidad()
print("--------------------") #decoración para que esté ordenadito
proyectoB.viabilidad()
print("-Prueba de Limite de Proyectos")
proyectoC = Proyecto("NuevoProducto", 50000.0)
proyectoD = Proyecto("Mantenimiento", 1000.0)
proyectoC.agregar_empleado(desarrollador1)
proyectoD.agregar_empleado(desarrollador1)
print("-Salarios calculados (despues de proyectos)")
procesar_empleados(empleados_general)
```

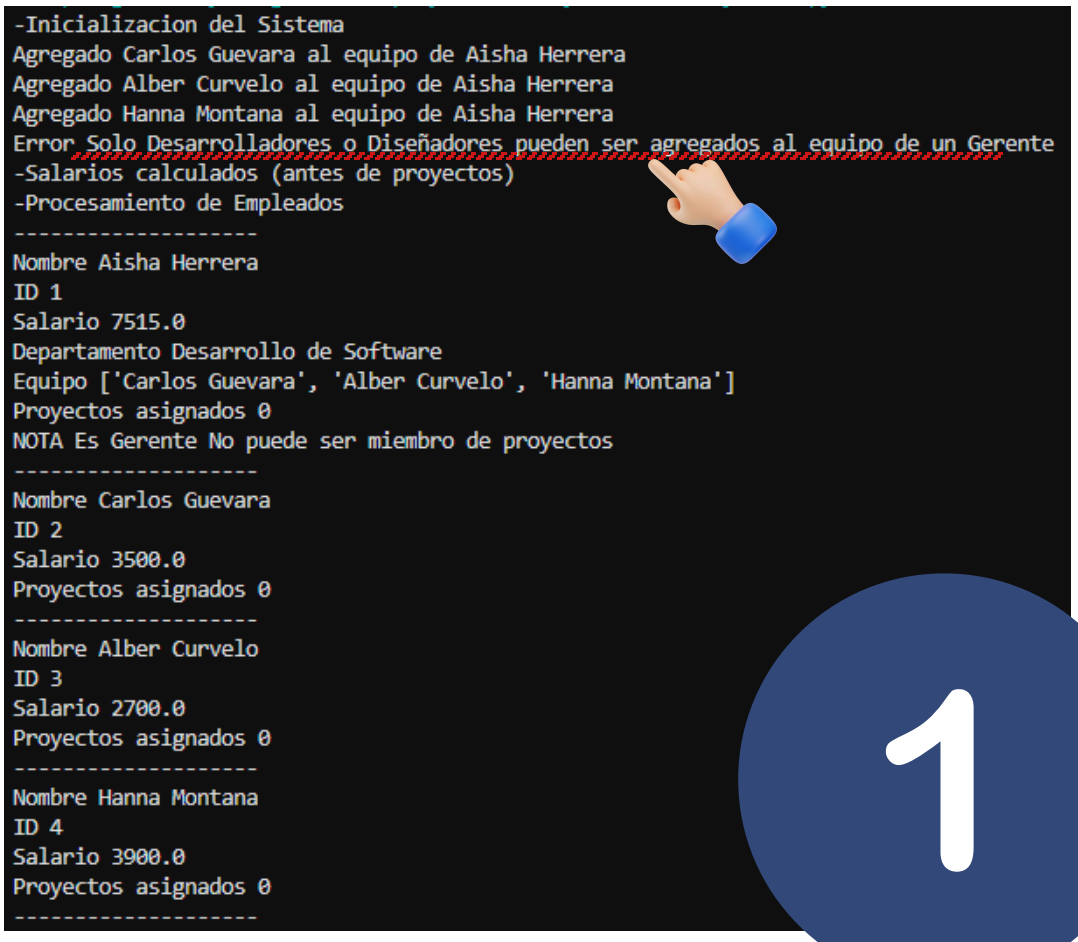
9

Resultados

Prueba realizada	Resultado esperado	Resultado real
Salario del Gerente	El salario del Gerente debe incluir el 15% de bono basado en el salario de su equipo (Desarrollador 1, Desarrollador 2, Diseñador 1).	Correcto. El salario base incrementa.
Límite de proyectos	El Desarrollador 1 (Carlos Guevara) tiene un límite de 3 proyectos. Al intentar asignarle el proyecto D (el cuarto), debe lanzar un error.	Correcto. Se capturó un ValueError con el mensaje: "Error El empleado Carlos Guevara excede su limite de 3 proyectos".
Asignación de Gerente	Intentar asignar al Gerente (Aisha Herrera) a un proyecto como miembro.	Correcto. Se captura un ValueError con el mensaje: "Error Los Gerentes no pueden ser asignados a proyectos como miembros".
Viabilidad de proyectos	El método viabilidad() calcula el costo total del proyecto (suma de salarios) y lo compara con el 70% del presupuesto.	Correcto. El proyecto A se valida (True) y el proyecto B no (False), demostrando que el cálculo es funcional.

10

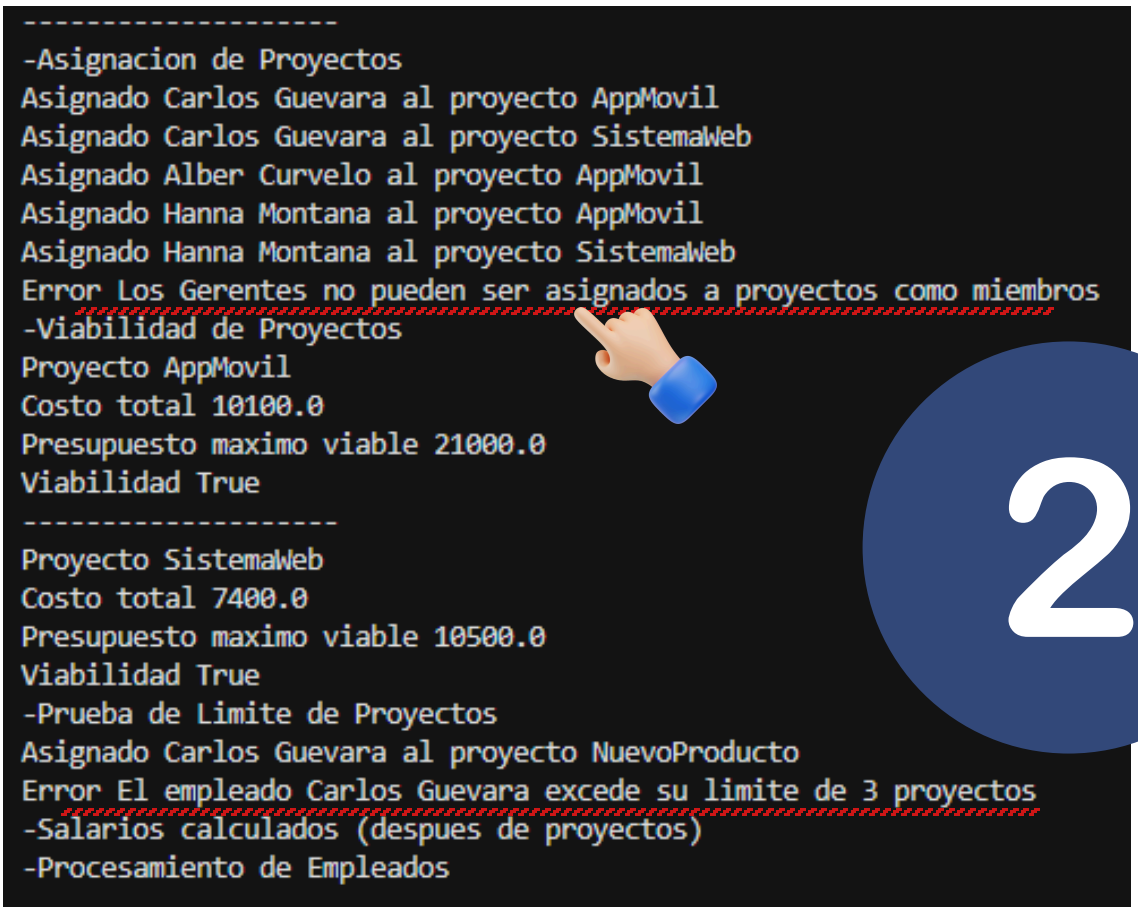
Evidencia fotográfica de resultados



1

11

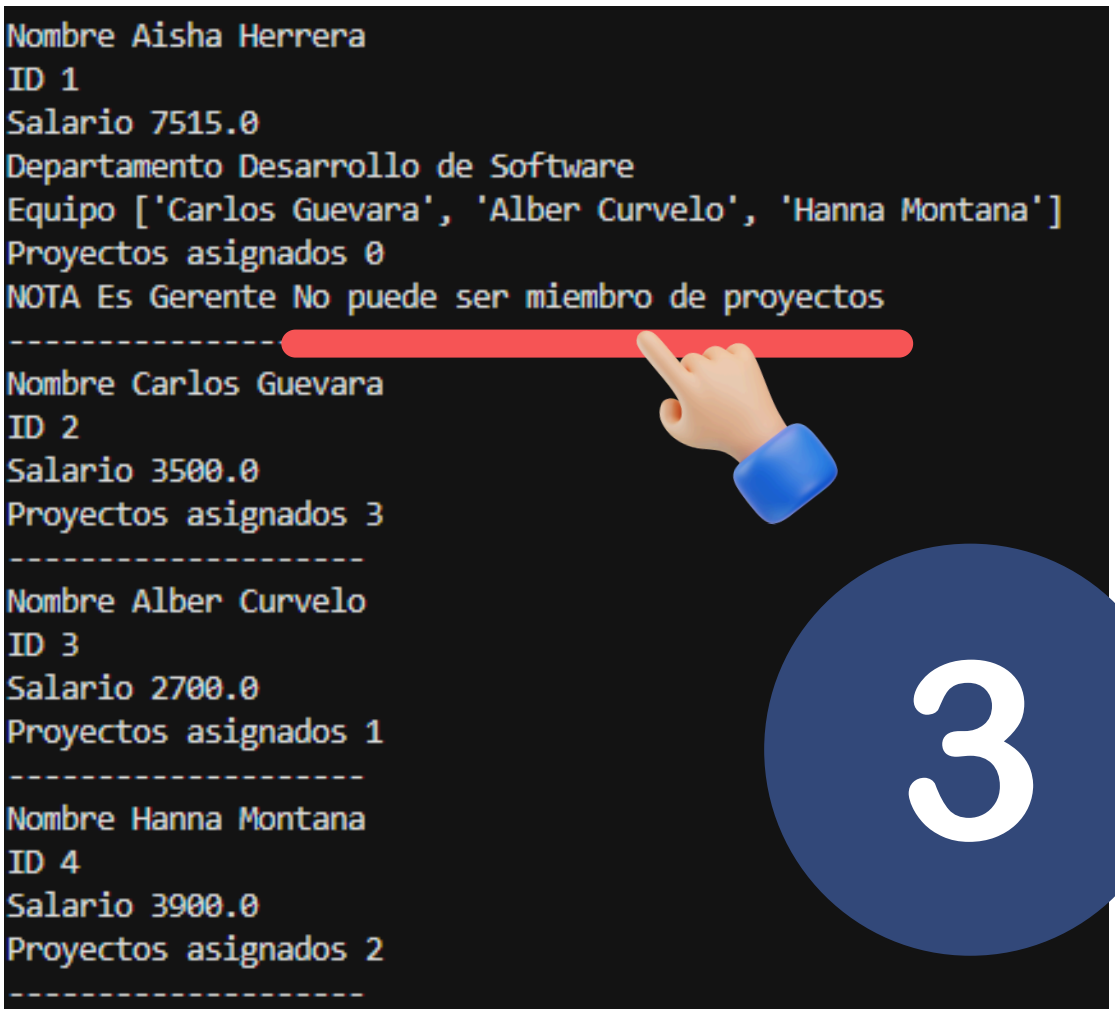
Evidencia fotográfica de resultados



2

12

Evidencia fotográfica de resultados



3

13

Conclusión

Fue posible implementar el sistema de manera satisfactoria, se cumplieron todos los requisitos y se llegó a los resultados esperados. Además de que se aplicaron los conceptos que fueron solicitados.

Enlace GitHub: https://github.com/naomyashley-cyber/Proyecto-Naomy.git