PNF Informática Sección 7122

Informe algoritmo

Proyecto 3: Programación orientada a objetos

Dhylan Castro 27671409

NOVIEMBRE 2025

Este proyecto busca crear un sistema que ayude a una empresa de tecnología a manejar mejor a su personal y los proyectos en los que trabajan. La idea es usar programación orientada a objetos para organizar todo de forma clara y práctica.

Dentro del sistema se definen tres tipos de empleados: desarrolladores, gerentes y diseñadores, cada uno con sus propias características y reglas para calcular su salario y para decidir en cuántos proyectos pueden participar. Además, existe la clase Proyecto, que sirve para llevar el control de quién está asignado, cuánto cuesta el equipo y si el presupuesto alcanza para hacerlo viable.

En pocas palabras, este trabajo sirve para practicar conceptos de programación y, al mismo tiempo, simular cómo sería gestionar personas y proyectos en una empresa real.

Clase Desarrollador:

```python
class Desarrollador(Empleado):
    def __init__(self, nombre, salario_base, lenguajes, nivel):
        super().__init__(nombre, salario_base)
        self.lenguajes = lenguajes
        self.nivel = nivel
        self.max_proyectos = 3

    def calcular_bono(self):
        if self.nivel == "Junior":
            return 200
        elif self.nivel == "SemiSenior":
            return 500
        elif self.nivel == "Senior":
            return 1000
        return 0

    def calcular_salario(self):
        return self._salario_base + self.calcular_bono()
```

Clase Gerente:

```python
class Gerente(Empleado):
    def __init__(self, nombre, salario_base, departamento):
        super().__init__(nombre, salario_base)
        self.departamento = departamento
        self.equipo = []
        self.max_proyectos = 0

    def calcular_bono(self):
        total_equipo = sum([empleado.calcular_salario() for empleado in self.equipo])
        return 0.15 * total_equipo

    def calcular_salario(self):
        return self._salario_base + self.calcular_bono()

    def agregar_al_equipo(self, empleado):
        if isinstance(empleado, (Desarrollador, Diseñador)):
            self.equipo.append(empleado)
        else:
            raise Exception("Solo se pueden agregar Desarrolladores o Diseñadores al equipo.")
```

Clase Diseñador:

```python
class Diseñador(Empleado):
    def __init__(self, nombre, salario_base, herramientas, especialidad):
        super().__init__(nombre, salario_base)
        self.herramientas = herramientas
        self.especialidad = especialidad
        self.max_proyectos = 2

    def calcular_bono(self):
        bono = 0
        if "Figma" in self.herramientas:
            bono += 300
        elif any(tool in self.herramientas for tool in ["Photoshop", "Illustrator"]) and len(self.herramientas) == 1:
            bono += 200
        if len(self.herramientas) >= 3:
            bono += 400
        return bono

    def calcular_salario(self):
        return self._salario_base + self.calcular_bono()
```

Clase Proyectos:

```python
class Proyecto:
    def __init__(self, nombre, presupuesto):
        self.nombre = nombre
        self.presupuesto = presupuesto
        self.empleados = []

    def agregar_empleado(self, empleado):
        if empleado in self.empleados:
            raise Exception(f"{empleado._nombre} ya está en el proyecto {self.nombre}.")
        if len(empleado.proyectos) >= empleado.max_proyectos:
            raise Exception(f"{empleado._nombre} excede su límite de proyectos.")
        self.empleados.append(empleado)

    def costo_total(self):
        return sum([empleado.calcular_salario() for empleado in self.empleados])

    def viabilidad(self):
        return self.costo_total() <= self.presupuesto * 0.7
```

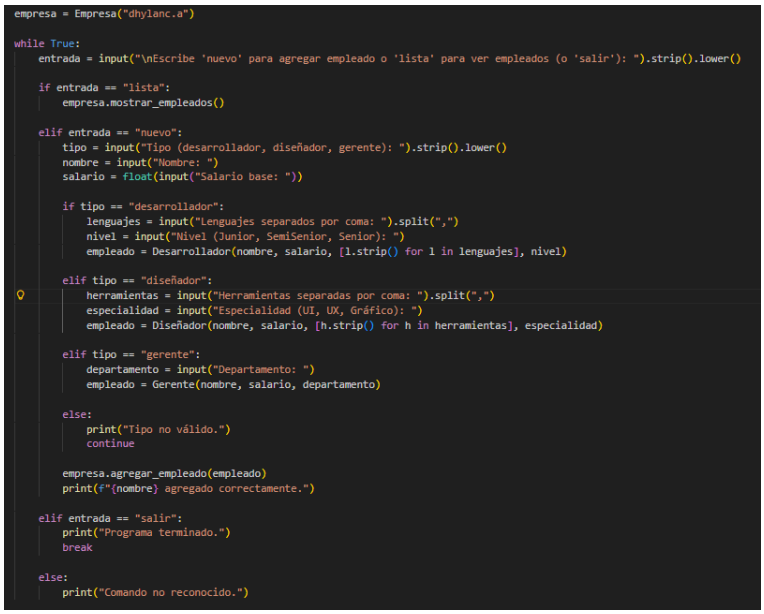
Clase Empresa:

```python
class Empresa:
    def __init__(self, nombre):
        self.nombre = nombre
        self.empleados = []

    def agregar_empleado(self, empleado):
        self.empleados.append(empleado)

    def mostrar_empleados(self):
        print(f"\nListado de empleados en {self.nombre}")
        print("-" * 60)
        for emp in self.empleados:
            info = emp.mostrar_informacion()
            print(f"Nombre: {info['Nombre']}")
            print(f"ID: {info['ID']}")
            print(f"Salario base: ${info['Salario base']}")
            print(f"Bono: ${info['Bono']}")
            print(f"Salario total: ${info['Salario total']}")
            print(f"Proyectos asignados: {info['Proyectos']}")
            print("-" * 60)
```

Agregar por consola:



```
empresa = Empresa("dhylanc.a")

while True:
    entrada = input("\nEscribe 'nuevo' para agregar empleado o 'lista' para ver empleados (o 'salir'): ").strip().lower()

    if entrada == "lista":
        empresa.mostrar_empleados()

    elif entrada == "nuevo":
        tipo = input("Tipo (desarrollador, diseñador, gerente): ").strip().lower()
        nombre = input("Nombre: ")
        salario = float(input("Salario base: "))

        if tipo == "desarrollador":
            lenguajes = input("Lenguajes separados por coma: ").split(",")
            nivel = input("Nivel (Junior, SemiSenior, Senior): ")
            empleado = Desarrollador(nombre, salario, [l.strip() for l in lenguajes], nivel)

        elif tipo == "diseñador":
            herramientas = input("Herramientas separadas por coma: ").split(",")
            especialidad = input("Especialidad (UI, UX, Gráfico): ")
            empleado = Diseñador(nombre, salario, [h.strip() for h in herramientas], especialidad)

        elif tipo == "gerente":
            departamento = input("Departamento: ")
            empleado = Gerente(nombre, salario, departamento)

        else:
            print("Tipo no válido.")
            continue

        empresa.agregar_empleado(empleado)
        print(f"{nombre} agregado correctamente.")

    elif entrada == "salir":
        print("Programa terminado.")
        break

    else:
        print("Comando no reconocido.")
```

este proyecto permite aplicar de forma práctica los conceptos de programación orientada a objetos para gestionar empleados y proyectos dentro de una empresa tecnológica. Con él se refuerzan conocimientos de herencia, polimorfismo y composición, mientras se simula un entorno real de organización y trabajo en equipo.

DHYLAN CASTRO
C.I:V-27.671.409
SECION 7122
ALGORITMO