Ejercicios Pseudocódigo y POO (Programación Orientada a Objetos)

Ejercicio 1: Banco y Cuentas Corrientes Diseñar un pseudocódigo para un programa orientado a objetos de un banco que gestiona cuentas corrientes.   
- Crea una clase Cliente con atributos para id, nombre y apellidos.   
- Crea una clase Movimiento con atributos para concepto y cantidad.   
- Crea una clase Cuenta con atributos para número, titular (id del usuario), movimientos y saldo. El saldo se actualiza a partir de los movimientos.   
- En el módulo principal, simula la creación de un cliente con dos cuentas corrientes y tres movimientos en cada cuenta. Muestra el saldo actual de cada cuenta.

**Solucion ejercicio1:**  
  
Clase Cliente

Atributos:

Id (autoincrementable)

nombre

apellidos

Métodos:

Constructor(id, nombre, apellidos)

this.id = id

this.nombre = nombre

this.apellidos = apellidos

Clase Movimiento

Atributos:

concepto

cantidad

Métodos:

Constructor(concepto, cantidad)

this.concepto = concepto

this.cantidad = cantidad

Clase Cuenta

Atributos:

numero

titularId

movimientos (lista de Movimiento)

saldo

Métodos:

Constructor(numero, titularId)

this.numero = numero

this.titularId = titularId

this.movimientos = []

this.saldo = 0

AgregarMovimiento(movimiento)

this.movimientos.agregar(movimiento)

this.saldo = this.saldo + movimiento.cantidad

ObtenerSaldo()

retornar this.saldo

Módulo Principal

// Creacion de cliente

cliente1 = nuevo Cliente(1, "Jimmer", "Carvajal")

// Creacion de cuentas para el cliente

cuenta1 = nuevo Cuenta(101, cliente1.id)

cuenta2 = nuevo Cuenta(102, cliente1.id)

// Creacion de movimientos para la cuenta 1

movimiento1\_1 = nuevo Movimiento("Ingreso", 500)

movimiento1\_2 = nuevo Movimiento("Retiro", -200)

movimiento1\_3 = nuevo Movimiento("Ingreso", 300)

cuenta1.AgregarMovimiento(movimiento1\_1)

cuenta1.AgregarMovimiento(movimiento1\_2)

cuenta1.AgregarMovimiento(movimiento1\_3)

// Creacion de movimientos para la cuenta 2

movimiento2\_1 = nuevo Movimiento("Ingreso", 1000)

movimiento2\_2 = nuevo Movimiento("Retiro", -150)

movimiento2\_3 = nuevo Movimiento("Ingreso", 200)

cuenta2.AgregarMovimiento(movimiento2\_1)

cuenta2.AgregarMovimiento(movimiento2\_2)

cuenta2.AgregarMovimiento(movimiento2\_3)

// Mostrar saldo actual de cada cuenta

Imprimir("Saldo de la cuenta 1: " + cuenta1.ObtenerSaldo())

Imprimir("Saldo de la cuenta 2: " + cuenta2.ObtenerSaldo())

Ejercicio 2: Tienda Online   
Diseñar un pseudocódigo para un sistema de tienda online con programación orientada a objetos.   
- Crea una clase Cliente con atributos para id, nombre y apellidos.   
- Crea una clase Artículo con atributos para código y precio.   
- Crea una clase Factura con atributos para número, cliente y líneas de factura.   
El total se calcula automáticamente.   
- En el módulo principal, representa una factura de un cliente que incluye varios artículos y muestra todos los datos de la factura, incluyendo el importe total.

**Solucion ejerc 2**

Clase Cliente

Atributos:

Id (autoincrementable)

nombre

apellidos

Métodos:

Constructor(id, nombre, apellidos)

this.id = id

this.nombre = nombre

this.apellidos = apellidos

Clase Artículo

Atributos:

codigo

precio

Métodos:

Constructor(codigo, precio)

this.codigo = codigo

this.precio = precio

Clase Factura

Atributos:

numero

cliente (objeto de Cliente)

lineasFactura (lista de Artículo)

Métodos:

Constructor(numero, cliente)

this.numero = numero

this.cliente = cliente

this.lineasFactura = []

AgregarArticulo(articulo)

this.lineasFactura.agregar(articulo)

CalcularTotal()

total = 0

Para cada articulo en this.lineasFactura

total = total + articulo.precio

retornar total

Módulo Principal

// Creacion de cliente

cliente1 = nuevo Cliente(1, "Jimmer", "Carvajal")

// Creacion de una factura para el cliente

factura1 = nuevo Factura(1001, cliente1)

// Creacion de artículos

articulo1 = nuevo Artículo("A001", 29.99)

articulo2 = nuevo Artículo("A002", 15.50)

articulo3 = nuevo Artículo("A003", 45.00)

// Agregando los artículos a la factura

factura1.AgregarArticulo(articulo1)

factura1.AgregarArticulo(articulo2)

factura1.AgregarArticulo(articulo3)

// Calcular el total de la factura

totalFactura = factura1.CalcularTotal()

// imprimimos la factura

Imprimir("Factura Número: " + factura1.numero)

Imprimir("Cliente: " + cliente1.nombre + " " + cliente1.apellidos)

Imprimir("Artículos en la factura:")

Para cada articulo en factura1.lineasFactura

Imprimir("Código: " + articulo.codigo + ", Precio: " + articulo.precio)

Fin Para

Imprimir("Total de la factura: " + totalFactura)

Ejercicio 3: Sistema de Biblioteca Diseñar un pseudocódigo para un sistema de biblioteca que gestione libros y préstamos.   
- Define una clase Libro con atributos para título, autor y disponibilidad.   
- Define una clase Usuario con atributos para id, nombre y lista de libros prestados.   
- Define una clase Préstamo con atributos para el libro, el usuario y las fechas de préstamo y devolución.   
- En el módulo principal, simula el préstamo de un libro a un usuario y luego su devolución. Recuerda actualizar la disponibilidad del libro.

**Solucion ejerc 3**

Clase Libro

Atributos:

titulo

autor

disponibilidad

Métodos:

Constructor(titulo, autor)

this.titulo = titulo

this.autor = autor

this.disponibilidad = verdadero // El libro está disponible al inicio

CambiarDisponibilidad(estado)

this.disponibilidad = estado

Clase Usuario

Atributos:

id

nombre

librosPrestados (lista de Libro)

Métodos:

Constructor(id, nombre)

this.id = id

this.nombre = nombre

this.librosPrestados = []

AgregarLibroPrestado(libro)

this.librosPrestados.agregar(libro)

RemoverLibroPrestado(libro)

this.librosPrestados.eliminar(libro)

Clase Préstamo

Atributos:

libro (objeto de Libro)

usuario (objeto de Usuario)

fechaPrestamo

fechaDevolucion

Métodos:

Constructor(libro, usuario, fechaPrestamo, fechaDevolucion)

this.libro = libro

this.usuario = usuario

this.fechaPrestamo = fechaPrestamo

this.fechaDevolucion = fechaDevolucion

Módulo Principal

// Crear un libro

libro1 = nuevo Libro("Cien años de soledad", "Gabriel García Márquez")

// Creando usuario

usuario1 = nuevo Usuario(1, "Jimmer Carvajal")

// Simular el préstamo de un libro

Si libro1.disponibilidad es verdadero Entonces

libro1.CambiarDisponibilidad(falso) // Cambiar disponibilidad a no disponible

usuario1.AgregarLibroPrestado(libro1) // Agregar libro a la lista de libros prestados

prestamo1 = nuevo Préstamo(libro1, usuario1, "2024-11-04", "") // Fecha de préstamo

Imprimir("Préstamo realizado: " + libro1.titulo + " a " + usuario1.nombre)

Sino

Imprimir("El libro " + libro1.titulo + " no está disponible para préstamo.")

// Simular la devolución del libro

libro1.CambiarDisponibilidad(true) // Cambiar disponibilidad a disponible

usuario1.RemoverLibroPrestado(libro1) // Remover libro de la lista de libros prestados

prestamo1.fechaDevolucion = "2024-11-10" // Fecha de devolución

Imprimir("Devolución realizada: " + libro1.titulo + " por " + usuario1.nombre)

Ejercicio 4: Gestión de Empleados Diseñar un pseudocódigo para un sistema que gestione empleados en una empresa.   
- Crea una clase Empleado con atributos para id, nombre, puesto y salario.   
- Crea una clase Departamento con atributos para código, nombre y lista de empleados.   
- Implementa métodos en la clase Departamento para agregar y remover empleados.   
- En el módulo principal, crea un departamento y añade varios empleados. Luego, muestra la información del departamento y de cada empleado.

**Solucion ejerc 4**

Clase Empleado

Atributos:

id

nombre

puesto

salario

Métodos:

Constructor(id, nombre, puesto, salario)

this.id = id

this.nombre = nombre

this.puesto = puesto

this.salario = salario

ObtenerInformacion()

retornar "ID: " + this.id + ", Nombre: " + this.nombre + ", Puesto: " + this.puesto + ", Salario: " + this.salario

Clase Departamento

Atributos:

codigo

nombre

empleados (lista de Empleado)

Métodos:

Constructor(codigo, nombre)

this.codigo = codigo

this.nombre = nombre

this.empleados = []

AgregarEmpleado(empleado)

this.empleados.agregar(empleado)

RemoverEmpleado(empleadoId)

Para cada empleado en this.empleados

Si empleado.id es igual a empleadoId Entonces

this.empleados.eliminar(empleado)

Imprimir("Empleado con ID " + empleadoId + " ha sido removido.")

retornar

Fin Si

Fin Para

Imprimir("Empleado con ID " + empleadoId + " no encontrado.")

MostrarInformacion()

Imprimir("Departamento: " + this.nombre + " (Código: " + this.codigo + ")")

Imprimir("Empleados en el departamento:")

Para cada empleado en this.empleados

Imprimir(empleado.ObtenerInformacion())

Fin Para

Módulo Principal

// Crear un departamento

departamento1 = nuevo Departamento(101, "Recursos Humanos")

// Crear empleados

empleado1 = nuevo Empleado(1, "Ana Gómez", "Gerente", 50000)

empleado2 = nuevo Empleado(2, "Carlos López", "Asistente", 30000)

empleado3 = nuevo Empleado(3, "María Pérez", "Analista", 40000)

// Agregar empleados al departamento

departamento1.AgregarEmpleado(empleado1)

departamento1.AgregarEmpleado(empleado2)

departamento1.AgregarEmpleado(empleado3)

// Mostrar información del departamento y de los empleados

departamento1.MostrarInformacion()

// Simular la remoción de un empleado

departamento1.RemoverEmpleado(2) // Remover al empleado con ID 2

// Mostrar información actualizada del departamento

departamento1.MostrarInformacion()