Examen Final Programación

Respuesta a las preguntas.

Fase 1: Análisis y Diseño del Sistema

1. ¿Qué clases principales crees que son necesarias para modelar este sistema? Justifica tus elecciones.
   Empleado: entidad del dominio con datos personales, categoría, salario y correo. Es central para el negocio.
   Proyecto: entidad del dominio con codigo, nombre y fechas. Es el otro eje del negocio.
   Asignación: clase de asociacion que modela la relacion muchos-a-muchos Empleado–Proyecto y guarda **fecha_asignacion**. Esto es lo recomendado cuando la relacion tiene datos propios.
   GestorSistema: orquesta las operaciones de crear/listar y de asignar/listar relaciones, separando responsabilidades de gestión respecto a los datos de las entidades (principio de responsabilidad única).

2. ¿Qué atributos y métodos deben incluirse en cada clase para cumplir con los requisitos planteados?
   Empleado
   Atributos (- privados):
   a. numero_carnet: string
   b. nombre: string
   c. fecha_nacimiento: string (YYYY-MM-DD)
   d. categoria: Categoria (enum)
   e. salario: double (por defecto 250000, rango [250000..500000])
   f. direccion: string (por defecto “San Jose”)
   g. telefono: string
   h. correo: string (unico; verificado globalmente)

   Metodos (+ publicos):
   i. constructores con/sin salario (validan edad >= 18, categoria valida, correo unico, salario en rango)
   j. getters/setters con validacion (setCorreo mantiene unicidad)
   k. mostrar(os): void (imprime todos los datos)

   Notas UML: visibilidad con + y -; metodos y atributos segun notacion estandar.
   Proyecto
   Atributos (-):
   l. codigo: string (unico)
   m. nombre: string (unico global, simulado con registro de nombres usados)
   n. fecha_inicio: string (YYYY-MM-DD)

o. fecha_finalizacion: string (YYYY-MM-DD)

Metodos (+):

p. constructor (valida unicidad de nombre)
q. getters/setters (setNombre valida unicidad)
r. mostrar(os): void

Asignacion (clase de asociacion Empleado–Proyecto)

Atributos (-):

s. carnet_empleado: string (FK logica a Empleado.numero_carnet)
t. codigo_proyecto: string (FK logica a Proyecto.codigo)
u. fecha_asignacion: string (YYYY-MM-DD, “hoy”)

Metodos (+):

v. constructor(carnet, codigo, fecha)
w. getters basicos

Regla: no duplicar (carnet_empleado, codigo_proyecto). La **clase de asociacion** se usa precisamente para almacenar informacion propia del vinculo en una relacion .

GestorSistema

Atributos (-):

x. empleados: vector<Empleado>
y. proyectos: vector<Proyecto>
z. asignaciones: vector<Asignacion>
aa. apoyo para unicidad (correos usados, nombres de proyecto usados)

Metodos (+):

bb. crearEmpleado(...) / listarEmpleados(os)
cc. crearProyecto(...) / listarProyectos(os)
dd. asignarEmpleadoAProyecto(carnet, codigo) // registra fecha actual; evita duplicados
ee. listarEmpleadosDeProyecto(codigo, os)
ff. listarProyectosDeEmpleado(carnet, os)

Razonamiento: **separa responsabilidades** (entidades con datos vs. servicio de gestion), lo cual reduce razones de cambio por clase.

3. Propón un diagrama de relación entre clases (puedes hacerlo en texto si no tienes herramientas gráficas).

class Empleado {

- numero_carnet: string
- nombre: string
- fecha_nacimiento: string

- categoria: Categoria
- salario: double
- direccion: string
- telefono: string
- correo: string
- Empleado(carnet, nombre, fecha_nac, categoria, direccion, telefono, correo)
- Empleado(carnet, nombre, fecha_nac, categoria, salario, direccion, telefono, correo)
- getCarnet(): string
- getNombre(): string
- getFechaNacimiento(): string
- getCategoria(): Categoria
- getSalario(): double
- getDireccion(): string
- getTelefono(): string
- getCorreo(): string
- setNombre(n: string): void
- setFechaNacimiento(f: string): void
- setCategoria(c: string): void
- setSalario(s: double): void
- setDireccion(d: string): void
- setTelefono(t: string): void
- setCorreo(c: string): void
- mostrar(os): void }

class Proyecto {

- codigo: string
- nombre: string
- fecha_inicio: string
- fecha_finalizacion: string
- Proyecto(codigo, nombre, fecha_inicio, fecha_fin)
- getCodigo(): string
- getNombre(): string
- getFechaInicio(): string
- getFechaFinalizacion(): string
- setNombre(n: string): void
- setFechaInicio(f: string): void
- setFechaFinalizacion(f: string): void
- mostrar(os): void }

class Asignacion {

- carnet_empleado: string
- codigo_proyecto: string
- fecha_asignacion: string
- Asignacion(carnet, codigo, fecha)
- getCarnetEmpleado(): string
- getCodigoProyecto(): string
- getFechaAsignacion(): string }

class GestorSistema {

- empleados: vector
- proyectos: vector
- asignaciones: vector
- correos_usados: vector
- nombres_proyecto_usados: vector
- crearEmpleado(...): bool
- listarEmpleados(os): void
- crearProyecto(...): bool
- listarProyectos(os): void
- asignarEmpleadoAProyecto(carnet, codigo): bool
- listarEmpleadosDeProyecto(codigo, os): void
- listarProyectosDeEmpleado(carnet, os): void }

Fase 5: Reflexión Final

1. ¿Qué decisiones de diseño tomaste durante el desarrollo y por qué?
   Clase de asociacion Asignacion para almacenar `fecha_asignacion` y evitar duplicados en una relacion muchos-a-muchos (Empleado–Proyecto).
   Separacion de responsabilidades: entidades (Empleado, Proyecto) solo contienen datos y validaciones propias; **GestorSistema** concentra crear/listar/asignar.
   Validaciones dentro de constructores/setters para garantizar reglas de negocio (edad, rango salarial, categoria valida, unicidad de correo y nombre).
   Unicidad simulada con vectores estaticos en cada clase (correos y nombres), lo que es suficiente para el alcance del examen.

2. ¿Qué aspectos del problema fueron más difíciles de modelar y cómo los resolviste?
   Modelar la relacion N–N con informacion propia (fecha). Se resolvio con una **clase de asociacion** (Asignacion) y verificando duplicados (carnet,codigo).
   Unicidad sin BD ni contenedores avanzados: se uso un vector estatico y comparaciones en minusculas para simular la restriccion.

3. Si tuvieras más tiempo, ¿qué mejorarías o agregarías al sistema?
   Validaciones extra: coherencia de fechas (fin >= inicio), formato estricto de telefono/correo.
   Indices mas eficientes (maps) y pruebas unitarias.
   Diagrama UML formal en herramienta y mayor detalle de multiplicidades/roles (por ejemplo, roles “empleado” y “proyecto” en Asignación). Lineamientos de estilo UML recomiendan explicitar multiplicidad y visibilidad para comunicar mejor. (Investigación)

Enlace Github

Iradcita/Examen-Final---Programacion-1