PROGRAMACIÓN II
SEGUNDA ENTREGA DEL TRABAJO PRÁCTICO
MODELADO DE TADs Y DIAGRAMA

Integrantes del grupo: Lobos, Aaron Alexander
Muñoz, Franco Tadeo

Docentes: José Nores - Miguel Gabrielli

Comisión: 03

Cuatrimestre - Año: Segundo Cuatrimestre - 2025

Sistema HomeSolution - Trabajo Práctico Parte 2
Ampliación del Problema y Modificaciones al Diseño

Nuevos Requerimientos
Tipos de Empleados
- Empleados Contratados (originales): Cobran por hora
- Empleados de Planta Permanente: Cobran por día, tienen categoría (INICIAL, TÉCNICO, EXPERTO), medio día cuenta como día completo, bonus 2% sin retrasos

Gestión de Retrasos
- Registrar cantidad de retrasos por empleado
- Dos estrategias de asignación: primer disponible o por menor cantidad de retrasos

Tareas Dinámicas
- Agregar tareas a proyectos en curso (actualiza fechas)
- Estados de tarea: terminada/no terminada
- Liberar empleado al terminar tarea

Historial
- Mantener historial de empleados por proyecto después de finalizar
- Extensibilidad para futuros tipos de empleados

TADs Modificados y Nuevos
TADs Nuevos:
- EmpleadoContratado (hereda/implementa concepto Empleado)
- EmpleadoPlanta (hereda/implementa concepto Empleado)
- HistorialProyecto (registro de empleados que trabajaron)

TADs Modificados:
- Tarea (agrega estado terminada)
- Empleado(implementa en concepto empleado)
- Proyecto (historial, agregar tareas dinámicamente)
- HomeSolution (estrategias asignación, consultas historial)

TAD Empleado(MODIFICADO)
Especificación
Valores (Atributos)
- nombre: String - Nombre del empleado
- legajo: int - Número identificatorio único
- disponible: boolean -Indica si está disponible para Asignación
- canticadRetrasos: int - Contador de veces que tuvo retrasos

IREP
- legajo debe ser único en el sistema y mayor que 0
- nombre no puede estar vacío
- la disponibilidad debe reflejar el estado de asignación actual
- cantidadRetrasos debe ser mayor o igual que 0

Estructuras de Datos

No se requiere una estructura de datos adicional. Representa dos tipos de empleados, uno que cobra por hora y uno que cobra por día

Interfaz pública

- Empleado(nombre: String, legajo: int,cantRetrasos: int): Empleado - crea un empleado
- estaDisponible(): abstract boolean - indica disponibilidad
- marcarComoAsignado(): abstract void - Marca como no disponible
- marcarComoDisponible: abstract void - Marca como disponible
- incrementarRetrasos(): abstract int - incrementa el contador de retrasos
- tuvoRetrasos(): abstract boolean - indica si tiene retrasos registrados
- verEmpleado(): void - devuelve si un empleado es contratado o de planta

TAD EmpleadoContratado
Especificación
Valores (Atributos)

- valorHora: double - Valor por hora de trabajo

IREP

- legajo debe ser único en el sistema y mayor que 0
- valorHora debe ser mayor que 0
- nombre no puede ser vacío
- cantidadRetrasos debe ser mayor o igual que 0
- La disponibilidad debe reflejar correctamente el estado de asignación actual

Estructuras de Datos
No requiere estructuras de datos adicionales. Representa a un empleado que cobra por hora.
Interfaz Pública
Operaciones

- crearEmpleadoContratado(valor: double): EmpleadoContratado - Crea empleado contratado con disponibilidad true y retrasos en 0
- estaDisponible(): boolean - Indica disponibilidad
- marcarComoAsignado(): void- Marca como no disponible
- marcarComoDisponible(): void- Marca como disponible
- incrementarRetrasos(): int - Aumenta contador de retrasos en 1
- calcularPago(dias: double): double - Calcula pago según días (0.5 = 4h, 1+ = 8h por día)
- tuvoRetrasos(): boolean - Indica si tiene retrasos registrados

TAD EmpleadoPlanta
Especificación
Valores (Atributos)

- valorDia: double - Valor por día completo de trabajo
- categoria: String - Categoría del empleado (INICIAL, TÉCNICO, EXPERTO)

IREP

- legajo debe ser único en el sistema y mayor que 0

- valorDia debe ser mayor que 0
- nombre no puede ser vacío
- categoria debe ser "INICIAL", "TÉCNICO" o "EXPERTO"
- cantidadRetrasos debe ser mayor o igual que 0
- La disponibilidad debe reflejar correctamente el estado de asignación actual

Estructuras de Datos
No requiere estructuras de datos adicionales. Representa a un empleado de planta permanente.

Interfaz Pública
Operaciones
- crearEmpleadoPlanta(valorD: double, cat: String): EmpleadoPlanta - Crea empleado de planta con disponibilidad true y retrasos en 0
- consultarCategoria(): String - Devuelve la categoría
- estaDisponible(): boolean - Indica disponibilidad
- consultarCantidadRetrasos(): int - Devuelve cantidad de retrasos
- marcarComoAsignado(): void- Marca como no disponible
- marcarComoDisponible(): void - Marca como disponible
- incrementarRetrasos(): int - Aumenta contador de retrasos en 1
- calcularPago(dias: double, sinRetrasos: boolean): double - Calcula pago, medio día cuenta como día completo, aplica bonus 2% si no hubo retrasos
- tuvoRetrasos(): boolean - Indica si tiene retrasos registrados

TAD HistorialProyecto
Especificación
Valores (Atributos)
- numeroProyecto: int - Número del proyecto al que pertenece
- empleadosAsignados: HashSet<Integer> - Conjunto de legajos que trabajaron en el proyecto
- tareasRealizadas: HashMap<Integer, ArrayList<Integer>>` - Mapeo legajo → lista de índices de tareas realizadas

IREP
- numeroProyecto debe ser mayor que 0 y corresponder a un proyecto existente
- Todos los legajos en empleadosAsignados deben haber trabajado efectivamente en el proyecto
- Para cada legajo en tareasRealizadas, debe estar también en empleadosAsignados
- Los índices de tareas en tareasRealizadas deben ser válidos para el proyecto

Estructuras de Datos
HashSet<Integer>: Almacena legajos únicos de empleados que trabajaron. Permite la verificación rápida de participación.
HashMap<Integer, ArrayList<Integer>>: Mapea cada empleado a las tareas que realizó. Permite consulta eficiente del historial detallado.

Interfaz Pública
Operaciones

- crearHistorial(numProy: int): HistorialProyecto - Inicializa historial vacío para un proyecto
- registrarEmpleadoEnTarea(legajo: int, indiceTarea: int): void - Agrega empleado y tarea al historial
- consultarEmpleadosDelProyecto(): ArrayList<Integer> - Devuelve lista de legajos que trabajaron
- consultarTareasDeEmpleado(legajo: int): ArrayList<Integer> - Devuelve índices de tareas realizadas por empleado
- participoEmpleado(legajo: int): boolean - Verifica si empleado trabajó en el proyecto

TAD Tarea (MODIFICADO)
Especificación
Valores (Atributos)

- titulo: String - Identificador de la tarea
- descripcion: String - Descripción detallada
- diasNecesarios: double - Días originalmente planificados
- diasRetraso: double - Días de retraso acumulados
- empleadoAsignado: int - Legajo del empleado asignado (-1 si no tiene)
- tieneAsignacion: boolean - Indica si tiene empleado asignado
- terminada: boolean - NUEVO: Indica si la tarea fue completada

IREP

- diasNecesarios debe ser mayor que 0, permite valores como 0.5 para medio día
- diasRetraso debe ser mayor o igual que 0
- Si tieneAsignacion es true, entonces empleadoAsignado debe ser > 0
- Si tieneAsignacion es false, entonces empleadoAsignado debe ser -1
- titulo no puede ser vacío
- descripcion no puede ser vacío
- Si terminada es true, entonces tieneAsignacion debe haber sido true en algún momento (históricamente)

Estructuras de Datos
No requiere estructuras de datos adicionales. Ahora almacena legajo del empleado en lugar del objeto completo.

Interfaz Pública
Operaciones (modificadas y nuevas)

- crearTarea(tit: String, desc: String, dias: double): Tarea - Crea tarea sin asignación, no terminada
- consultarLegajoAsignado(): int - Devuelve legajo asignado o -1
- asignarEmpleado(legajo: int): void - Asigna empleado por legajo
- registrarRetraso(dias: double): double - Suma días de retraso
- tieneEmpleadoAsignado(): boolean - Indica si tiene empleado
- estaTerminada(): boolean - NUEVO: Indica si fue completada
- marcarComoTerminada(): void - NUEVO: Marca tarea como terminada
- tuvoRetraso(): boolean - NUEVO: Indica si tiene retrasos > 0

TAD Proyecto (MODIFICADO)
Especificación

Valores (Atributos)

- numero: int - Identificador único
- cliente: Cliente - Datos del cliente
- direccion: String - Dirección de la vivienda
- tareas: ArrayList<Tarea> - Lista de tareas
- fechaInicio: int - Fecha de inicio (AAAAMMDD)
- fechaEstimadaFin: int - Fecha estimada de finalización
- fechaRealFin: int - Fecha real de finalización
- finalizado: boolean - Indica si está terminado
- costoCalculado: double - NUEVO: Costo precalculado al crear/modificar proyecto
- historial: HistorialProyecto - NUEVO: Registro de empleados que trabajaron

IREP

- numero debe ser único y mayor que 0
- fechaInicio ≤ fechaEstimadaFin ≤ fechaRealFin
- Si finalizado es true, todas las tareas deben estar terminadas
- tareas no puede estar vacía
- direccion no puede ser vacía
- costoCalculado debe ser mayor que 0
- Si una tarea está terminada, su empleado debe estar en el historial

Estructuras de Datos
ArrayList<Tarea>: Permite acceso O(1) por índice, agregar tareas dinámicamente al final.
HistorialProyecto: Mantiene registro de todos los empleados que trabajaron incluso después de finalizar.

Interfaz Pública
Operaciones (modificadas y nuevas)

- crearProyecto(num: int, cli: Cliente, dir: String, ts: ArrayList<Tarea>, fIni: int, fEst: int): Proyecto - Crea proyecto con costo precalculado
- agregarTarea(tit: String, desc: String, dias: double): void- MODIFICADO: Agrega tarea, actualiza fechas y recalcula costo
- asignarEmpleadoATarea(indice: int, legajo: int): void - Asigna y registra en historial
- registrarRetrasoEnTarea(indice: int, dias: double, legajo: int): void - Registra retraso, actualiza fechaRealFin y recalcula costo
- finalizarTarea(indice: int): void - NUEVO: Marca tarea terminada, libera empleado
- finalizar(fecha: int): void- Marca proyecto finalizado, libera todos los empleados
- estaFinalizado(): boolean - Indica si está terminado
- estaPendiente(): boolean - Indica si hay tareas sin asignar
- estaActivo(): boolean - NUEVO: No finalizado y todas las tareas asignadas
- hayRetrasos(): boolean - Verifica retrasos
- cantidadTareas(): int - Cantidad de tareas

TAD HomeSolution (MODIFICADO)
Especificación
Valores (Atributos)

- empleados: HashMap<Integer, Empleado> - Mapeo legajo → empleado
- proyectos: HashMap<Integer, Proyecto> - Mapeo número → proyecto

- asignacionesPorTarea: HashMap<Tupla<Integer, Integer>, Integer>` - (proyecto, tarea) → legajo
- proyectosFinalizados: HashSet<Integer> - Proyectos terminados
- proyectosPendientes: HashSet<Integer> - Proyectos con tareas sin asignar
- proyectosActivos: HashSet<Integer> - NUEVO: Proyectos en progreso (todas tareas asignadas, no finalizados)

IREP

- Todos los legajos en empleadosContratados y empleadosPlanta deben ser únicos entre sí y mayores que 0
- Un legajo no puede estar en ambos diccionarios de empleados
- Todos los números en proyectos deben ser únicos y mayores que 0
- Para cada legajo en asignacionesPorEmpleado, debe existir en empleadosContratados o empleadosPlanta
- Para cada (proyecto, tarea) en asignacionesPorTarea, el proyecto y el índice deben ser válidos
- proyectosFinalizados ∩ proyectosPendientes = ∅
- proyectosFinalizados ∩ proyectosActivos = ∅
- proyectosPendientes ∩ proyectosActivos = ∅ (mutuamente excluyentes)
- Consistencia de mapeos dobles entre asignacionesPorEmpleado y asignacionesPorTarea
- Si empleado no disponible, debe tener al menos una asignación activa (tarea no terminada)

Estructuras de Datos
HashMap<Integer, Proyecto>: Proyectos con costo precalculado, acceso O(1) para consulta de costo.

HashMap<Tupla<Integer, Integer>, Integer>: Asignaciones por tarea para reasignación O(1).
HashSet<Integer> (proyectosActivos): Proyectos en curso con todas tareas asignadas, consulta O(1).

Interfaz Pública
Operaciones (modificadas y nuevas)
Gestión de Empleados

- registrarEmpleado(nom: String, leg: int, cantRetrasos: int): void - NUEVO: Registra empleado

Gestión de Proyectos

- registrarProyecto(num: int, cli: Cliente, dir: String, ts: ArrayList<Tarea>, fIni: int, fEst: int): void- Registra proyecto con costo precalculado
- agregarTareaAProyecto(numProyecto: int, tit: String, desc: String, dias: double): void- NUEVO: Agrega tarea dinámica, actualiza fechas y costo

Asignación de Empleados

- asignarEmpleadoTarea(numProyecto: int, indiceTarea: int): void - Asigna primer disponible
- asignarEmpleadoMenosRetrasos(numProyecto: int, indiceTarea: int): void- NUEVO: Asigna por menor cantidad de retrasos

Gestión de Tareas

- registrarRetraso(numProyecto: int, indiceTarea: int, dias: double): void- Registra retraso, actualiza contador empleado y recalcula costo
- finalizarTarea(numProyecto: int, indiceTarea: int): void- NUEVO: Marca tarea terminada, libera empleado

Finalización

- finalizarProyecto(numProyecto: int, fecha: int): void- Finaliza proyecto, libera empleados

Reasignación

- reasignarEmpleado(numProyecto: int, indiceTarea: int, nuevoLegajo: int): void- Reasigna en O(1)
- reasignarEmpleadoEficiente(numProyecto: int, indiceTarea: int): void- NUEVO: Reasigna buscando menor cantidad de retrasos

Consultas de Costos

- consultarCostoProyecto(numProyecto: int): double - MODIFICADO: Devuelve costo en O(1) desde proyecto

Informes de Proyectos

- informarProyectosFinalizados(): ArrayList<Integer> - NUEVO: Lista de proyectos finalizados
- informarProyectosPendientes(): ArrayList<Integer> - Lista de proyectos pendientes
- informarProyectosActivos(): ArrayList<Integer> - NUEVO: Lista de proyectos activos
- consultarEstadoProyecto(numProyecto: int): boolean - Devuelve estado en O(1)

Consultas de Empleados

- informarEmpleadosNoAsignados(): ArrayList<Integer> - NUEVO: Lista legajos de empleados disponibles
- consultarEmpleadoTuvoRetrasos(legajo: int): boolean - NUEVO: Verifica si empleado tuvo retrasos
- consultarEmpleadosAsignadosProyecto(numProyecto: int): ArrayList<Integer> - NUEVO: Lista empleados activos en proyecto

Justificación de Complejidades

O(1) para Consulta de Costo:

El costo se calcula al crear/modificar el proyecto y se almacena en costoCalculado. La consulta es un simple acceso al atributo.

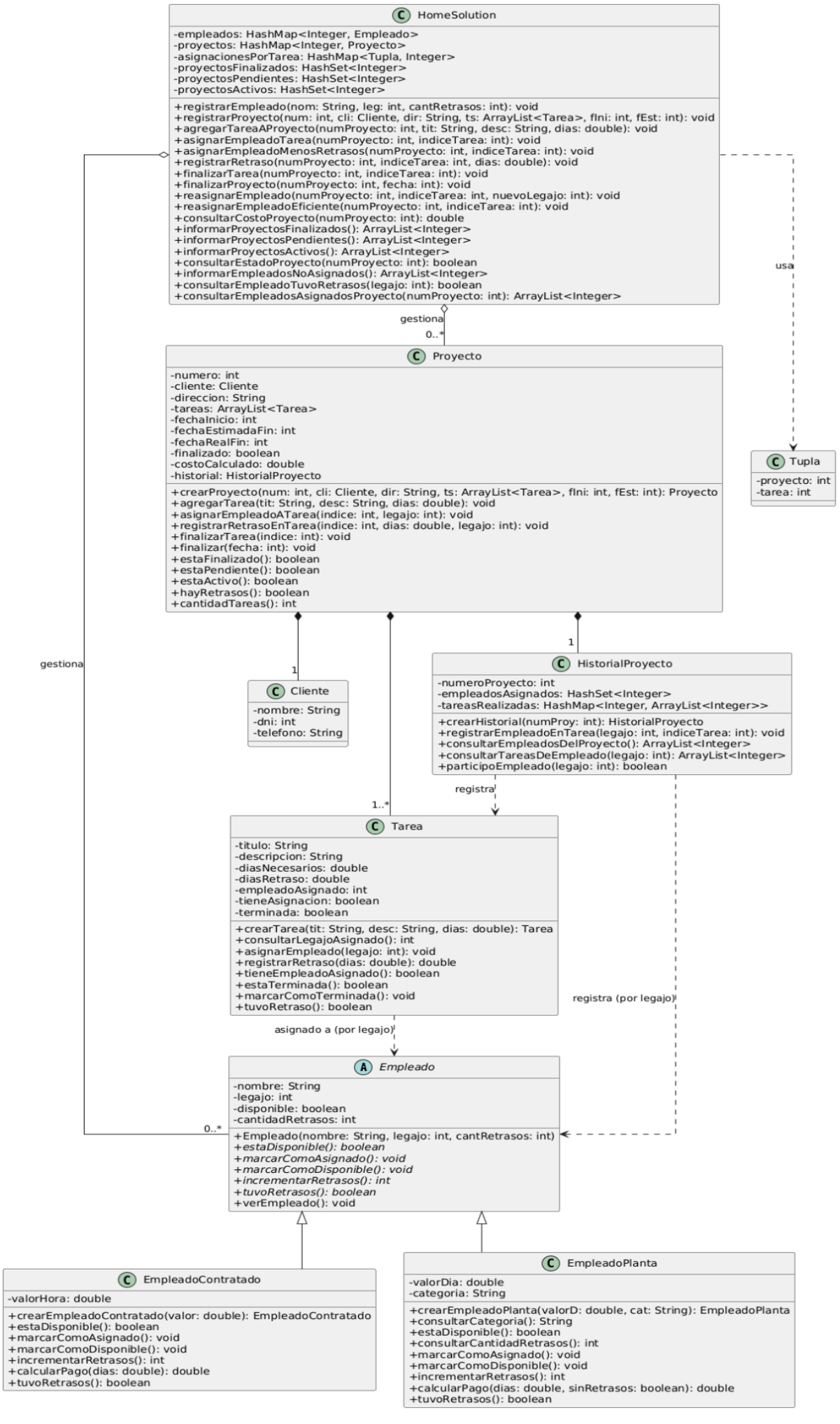
O(1) para Reasignación:

Mismo mecanismo de doble mapeo de la Parte 1, con actualización adicional del contador de retrasos si corresponde.

Asignación por Menor Retrasos:

Requiere recorrer empleados disponibles O(n) donde n = cantidad de empleados, pero es la forma más eficiente de encontrar el óptimo sin estructuras adicionales complejas.

Diagrama de Clases


HomeSolution
-empleados: HashMap<Integer, Empleado>
-proyectos: HashMap<Integer, Proyecto>
-asignacionesPorTarea: HashMap<Tupla, Integer>
-proyectosFinalizados: HashSet<Integer>
-proyectosPendientes: HashSet<Integer>
-proyectosActivos: HashSet<Integer>
+registrarEmpleado(nom: String, leg: int, cantRetrasos: int): void
+registrarProyecto(num: int, cli: Cliente, dir: String, ts: ArrayList<Tarea>, fIni: int, fEst: int): void
+agregarTareaAProyecto(numProyecto: int, tit: String, desc: String, dias: double): void
+asignarEmpleadoTarea(numProyecto: int, indiceTarea: int): void
+asignarEmpleadoMenosRetrasos(numProyecto: int, indiceTarea: int): void
+registrarRetraso(numProyecto: int, indiceTarea: int, dias: double): void
+finalizarTarea(numProyecto: int, indiceTarea: int): void
+finalizarProyecto(numProyecto: int, fecha: int): void
+reasignarEmpleado(numProyecto: int, indiceTarea: int, nuevoLegajo: int): void
+reasignarEmpleadoEficiente(numProyecto: int, indiceTarea: int): void
+consultarCostoProyecto(numProyecto: int): double
+informarProyectosFinalizados(): ArrayList<Integer>
+informarProyectosPendientes(): ArrayList<Integer>
+informarProyectosActivos(): ArrayList<Integer>
+consultarEstadoProyecto(numProyecto: int): boolean
+informarEmpleadosNoAsignados(): ArrayList<Integer>
+consultarEmpleadoTuvoRetrasos(legajo: int): boolean
+consultarEmpleadosAsignadosProyecto(numProyecto: int): ArrayList<Integer>
usa
gestiona
0..*
Proyecto
-numero: int
-cliente: Cliente
-direccion: String
-tareas: ArrayList<Tarea>
-fechaInicio: int
-fechaEstimadaFin: int
-fechaRealFin: int
-finalizado: boolean
-costoCalculado: double
-historial: HistorialProyecto
+crearProyecto(num: int, cli: Cliente, dir: String, ts: ArrayList<Tarea>, fIni: int, fEst: int): Proyecto
+agregarTarea(tit: String, desc: String, dias: double): void
+asignarEmpleadoATarea(indice: int, legajo: int): void
+registrarRetrasoEnTarea(indice: int, dias: double, legajo: int): void
+finalizarTarea(indice: int): void
+finalizar(fecha: int): void
+estaFinalizado(): boolean
+estaPendiente(): boolean
+estaActivo(): boolean
+hayRetrasos(): boolean
+cantidadTareas(): int
Tupla
-proyecto: int
-tarea: int
gestiona
1
Cliente
-nombre: String
-dni: int
-telefono: String
1..*
1
HistorialProyecto
-numeroProyecto: int
-empleadosAsignados: HashSet<Integer>
-tareasRealizadas: HashMap<Integer, ArrayList<Integer>>
+crearHistorial(numProy: int): HistorialProyecto
+registrarEmpleadoEnTarea(legajo: int, indiceTarea: int): void
+consultarEmpleadosDelProyecto(): ArrayList<Integer>
+consultarTareasDeEmpleado(legajo: int): ArrayList<Integer>
+participoEmpleado(legajo: int): boolean
registra
Tarea
-titulo: String
-descripcion: String
-diasNecesarios: double
-diasRetraso: double
-empleadoAsignado: int
-tieneAsignacion: boolean
-terminada: boolean
+crearTarea(tit: String, desc: String, dias: double): Tarea
+consultarLegajoAsignado(): int
+asignarEmpleado(legajo: int): void
+registrarRetraso(dias: double): double
+tieneEmpleadoAsignado(): boolean
+estaTerminada(): boolean
+marcarComoTerminada(): void
+tuvoRetraso(): boolean
registra (por legajo)
asignado a (por legajo)
Empleado
-nombre: String
-legajo: int
-disponible: boolean
-cantidadRetrasos: int
+Empleado(nombre: String, legajo: int, cantRetrasos: int)
+estaDisponible(): boolean
+marcarComoAsignado(): void
+marcarComoDisponible(): void
+incrementarRetrasos(): int
+tuvoRetrasos(): boolean
+verEmpleado(): void
0..*
EmpleadoContratado
-valorHora: double
+crearEmpleadoContratado(valor: double): EmpleadoContratado
+estaDisponible(): boolean
+marcarComoAsignado(): void
+marcarComoDisponible(): void
+incrementarRetrasos(): int
+calcularPago(dias: double): double
+tuvoRetrasos(): boolean
EmpleadoPlanta
-valorDia: double
-categoria: String
+crearEmpleadoPlanta(valorD: double, cat: String): EmpleadoPlanta
+consultarCategoria(): String
+estaDisponible(): boolean
+consultarCantidadRetrasos(): int
+marcarComoAsignado(): void
+marcarComoDisponible(): void
+incrementarRetrasos(): int
+calcularPago(dias: double, sinRetrasos: boolean): double
+tuvoRetrasos(): boolean

Extensibilidad para Nuevos Tipos de Empleados
El diseño usa herencia/polimorfismo con clase abstracta Empleado:

- Nuevos tipos solo implementan calcularPago() con su lógica específica
- HomeSolution puede agregar nuevo HashMap para el tipo
- Las operaciones de asignación funcionan con cualquier tipo mediante polimorfismo
- Impacto mínimo: Solo agregar nuevo HashMap y método de registro en HomeSolution