Universidad del Valle de Guatemala

Nathalie Valle – 251290

Análisis - Ejercicio 3

1. **Requisitos funcionales del sistema**
    1. Registrar miembros del gimnasio: El sistema debe permitir almacenar los datos personales de cada miembro, como nombre, ID y tipo de membresía. Esto resuelve el problema de tener información dispersa y facilita el acceso rápido a los datos.
    2. Registrar entrenadores: Se debe poder registrar a los entrenadores que trabajan en el gimnasio, incluyendo su nombre e ID y vincularlos con los miembros que atienden. Esto permite saber qué entrenador está a cargo de cada persona y evitar sobrecargas.
    3. Registrar rutinas de ejercicio: El sistema debe permitir crear rutinas con nombre, descripción y estado (activa/inactiva), para poder asignarlas a los miembros. Esto soluciona la confusión sobre qué ejercicios debe seguir cada socio.
    4. Asignar entrenadores a miembros: Cada miembro debe tener un entrenador asignado. Esto permite distribuir la carga de trabajo y saber quién está atendiendo a quién.
    5. Asignar rutinas a miembros: Los miembros deben tener una o varias rutinas asignadas. Esto permite personalizar el entrenamiento y llevar control de qué ejercicios realiza cada persona.
    6. Consultar número total de miembros inscritos: El sistema debe poder mostrar cuántas personas están activas en el gimnasio. Esto ayuda a la administración a tener una visión clara del tamaño de la comunidad.
    7. Consultar tipo de membresía de cada miembro: Saber qué tipo de membresía tiene cada persona permite gestionar pagos, promociones y vencimientos.
    8. Consultar rutinas activas y cuántas hay: El sistema debe permitir saber cuántas rutinas están en uso actualmente. Esto ayuda a evaluar la variedad y vigencia de los entrenamientos.
    9. Consultar qué entrenador atiende a cada miembro: Esta funcionalidad permite verificar la asignación de entrenadores y detectar sobrecargas.
    10. Identificar la rutina con más practicantes: Esta métrica permite saber qué rutinas son más populares y podrían necesitar ajustes o más recursos.
    11. Identificar el entrenador con más alumnos: Esta funcionalidad permite detectar desequilibrios en la carga de trabajo y tomar decisiones para redistribuir miembros.
    12. Permitir crecimiento dinámico: El sistema debe usar estructuras dinámicas para que el número de miembros, entrenadores y rutinas pueda crecer sin límite fijo.
2. **Clases y propósito**
    1. Clase Miembro: Representa a una persona inscrita en el gimnasio. Esta clase modela los datos personales del socio y su relación con entrenadores y rutinas. Es esencial para centralizar la información de cada usuario del gimnasio.

2. Clase Entrenador: Modela a los entrenadores que trabajan en el gimnasio. Permite registrar sus datos y llevar control de los miembros que atienden. Es clave para evitar sobrecargas y distribuir el trabajo de forma equitativa.
3. Clase Rutina: Representa una rutina de ejercicios. Incluye nombre, descripción, estado (activa/inactiva) y los miembros que la practican. Esta clase permite organizar los entrenamientos y analizar su popularidad.
4. Clase Gimnasio: Es la clase central que gestiona todas las listas dinámicas de miembros, entrenadores y rutinas. Actúa como controlador del sistema, permitiendo agregar, consultar y generar reportes. Es el núcleo de la lógica del programa.
5. Clase Main: Es la única clase que interactúa directamente con el usuario. Se encarga de mostrar mensajes en consola y recopilar datos. Su propósito es mantener la separación entre la lógica del sistema y la interfaz textual.

3. Atributos y propósito

1. Miembro:
    - Nombre: Identifica al socio por su nombre completo.
    - ID: Permite distinguir a cada miembro de forma única.
    - Membresía: Indica el tipo de contrato (mensual, anual, etc.), útil para gestión administrativa.
    - Entrenador: Referencia al entrenador asignado, permite saber quién lo atiende.
    - Rutinas: Lista de rutinas asignadas, permite personalizar el entrenamiento.
2. Entrenador:
    - Nombre: Identifica al entrenador.
    - ID: Identificador único para evitar confusiones.
    - Miembros: Lista de miembros que atiende, permite saber su carga de trabajo.
3. Rutina:
    - Nombre: Identifica la rutina.
    - ID: Identificador único.
    - Descripción: Explica brevemente los ejercicios incluidos.
    - Miembros: Lista de miembros que la practican, útil para medir popularidad.
    - Activa: Indica si la rutina está en uso actualmente.
4. Gimnasio:
    - Miembros: Lista dinámica de todos los socios registrados.
    - Entrenadores: Lista dinámica de entrenadores.
    - Rutinas: Lista dinámica de rutinas disponibles.
5. Main:
    - Gimnacio: gimnasio

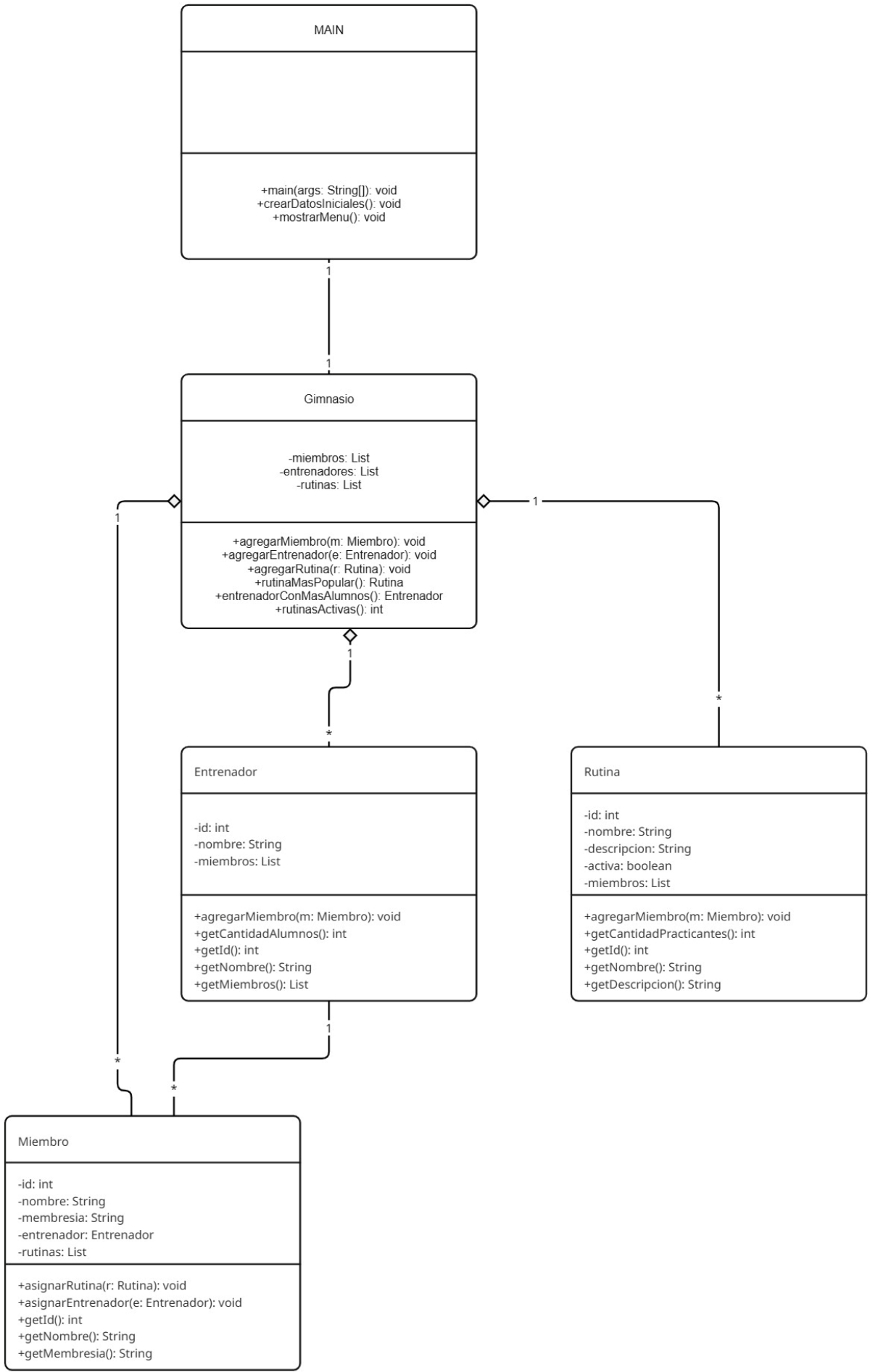
4. Métodos y propósito

6. Miembro:

- asignarRutina(Rutina r): Añade una rutina a la lista del miembro. Permite personalizar su entramiento.
- asiganarEntrenador(Entrenador e): Asocia un entrenador al miembro. Permite saber quién lo atiende.

7. Entrenador:
    - agregarMiembro(Miembro m): Añade un miembro a su lista. Permite llevar control de su carga de trabajo.
    - getCantidadAlumnos(): Devuelve el número de miembros que atiende. Útil para reportes.
8. Rutina:
    - agregarMiembro(Miembro m): Añade un miembro a la rutina. Permite saber quién la practica.
    - getCantidadPracticantes(): Devuelve cuántos miembros la practican. Útil para medir popularidad.
9. Gimnasio:
    - agregarMiembro(Miembro m): Añade un nuevo socio.
    - agregarEntrenaodr(Entrenador e)): Añade un nuevo entrenador.
    - agregarRutina(Rutina r): Añade una nueva rutina.
    - rutinaMasPopular(): Devuelve la rutina con más practicantes.
    - enntrenadorConMasAlumnos(): Devuelve el entrenador con más miembros asignados.
    - rutinasActivas(): Devuelve el número de rutinas activas.
10. Main:
    - Main(String[] args): void: punto de entrada
    - crearDatosInciales(): void: crea entrenadores, miembros y rutina prueba
    - mostrarMenu():. Void: muestra opciones al usuario para consultar estadísticas.

Diseño


MAIN
+main(args: String[]): void
+crearDatosIniciales(): void
+mostrarMenu(): void
1
1
Gimnasio
-miembros: List
-entrenadores: List
-rutinas: List
+agregarMiembro(m: Miembro): void
+agregarEntrenador(e: Entrenador): void
+agregarRutina(r: Rutina): void
+rutinaMasPopular(): Rutina
+entrenadorConMasAlumnos(): Entrenador
+rutinasActivas(): int
1
1
1
*
*
Entrenador
-id: int
-nombre: String
-miembros: List
+agregarMiembro(m: Miembro): void
+getCantidadAlumnos(): int
+getId(): int
+getNombre(): String
+getMiembros(): List
Rutina
-id: int
-nombre: String
-descripcion: String
-activa: boolean
-miembros: List
+agregarMiembro(m: Miembro): void
+getCantidadPracticantes(): int
+getId(): int
+getNombre(): String
+getDescripcion(): String
1
*
*
Miembro
-id: int
-nombre: String
-membresia: String
-entrenador: Entrenador
-rutinas: List
+asignarRutina(r: Rutina): void
+asignarEntrenador(e: Entrenador): void
+getId(): int
+getNombre(): String
+getMembresia(): String